CNN for MNIST

MNIST Database

- https://github.com/amaas/stanford_dl_ex/tree/master/common
- 70,000 images of handwritten numbers
- In general, 60,0000 images for training and 10,000 for validation
- Use **loadMNISTImage.m** & **loadMNISTLabels.m**

Figure 6-17. A 28-by-28 pixel black-and-white image from the MNIST database

CNN Architecture

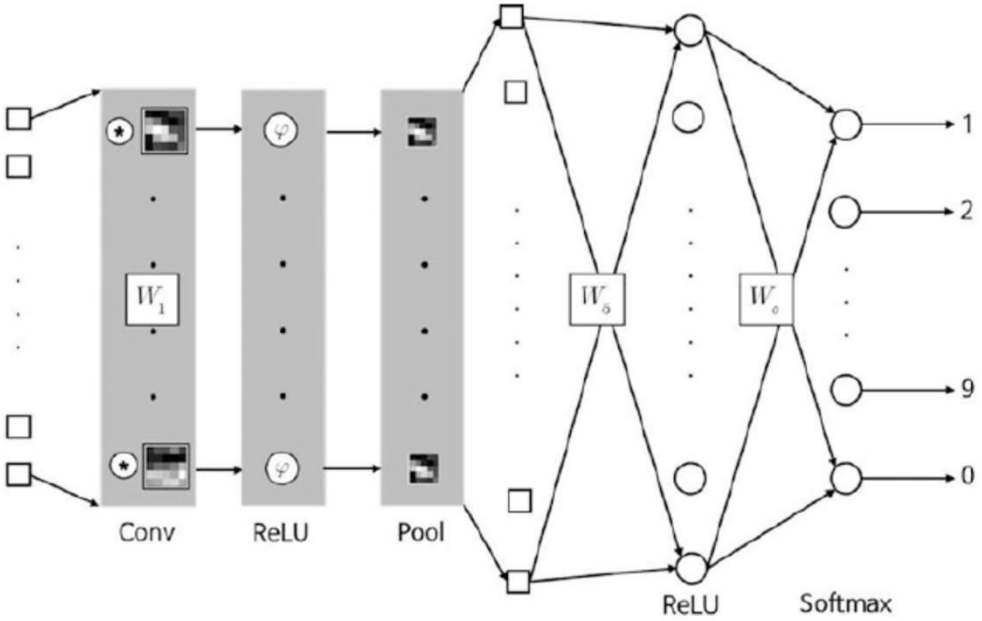


Figure 6-18. The architecture of this neural network

Layer	Remark	Activation Function
Input	28×28 nodes	-
Convolution	20 convolution filters (9×9)	ReLU
Pooling	1 mean pooling (2×2)	-
Hidden	100 nodes	ReLU
Output	10 nodes	Softmax

CNN Structure

- MNISTConv.m
  - Learning or training
    - Minibatches
    - Conv-Pooling-ReLU-Softmax
    - Backpropagation
      - Backward pooling
      - Backward convolution
    - Update weights with moments
  - Test or inferencing by TestMNISTConv.m
  - PlotFeatrues.m

Mini-batch

```
bsize = 100;
blist = 1:bsize:(N-bsize+1);

for batch = 1:length(blist)
  ...
  begin = blist(batch);
  for k = begin:begin+bsize-1
    ...
    dW1 = dW1 + delta2_x;
    dW5 = dW5 + delta5*y4';
    dWo = dWo + delta *y5';
  end
  dW1 = dW1 / bsize;
  dW5 = dW5 / bsize;
  dWo = dWo / bsize;
  ...
end
```

```
blist = [ 1, 101, 201, 301, ..., 7801, 7901 ]
```

Forward Propagation

```
% Forward pass = inference
%
x  = X(:, :, k);              % Input,           28x28
y1 = Conv(x, W1);             % Convolution,  20x20x20
y2 = ReLU(y1);                %
y3 = Pool(y2);                % Pool,         10x10x20
y4 = reshape(y3, [], 1);      %                  2000
v5 = W5*y4;                   % ReLU,             360
y5 = ReLU(v5);                %
v  = Wo*y5;                   % Softmax,           10
y  = Softmax(v);              %
```

One-hot Encoding

```
% One-hot encoding
%
d = zeros(10, 1);
d(sub2ind(size(d), D(k), 1)) = 1;
```

Note

sub2ind

Single index from subscripts

Syntax

```
IND = sub2ind(siz,I,J)
IND = sub2ind(siz,I1,I2,...,In)
```

Description

The `sub2ind` command determines the equivalent single index corresponding to a set of subscript values.

`IND = sub2ind(siz,I,J)` returns the linear index equivalent to the row and column subscripts `I` and `J` for a matrix of size `siz`. `siz` is a 2-element vector, where `siz(1)` is the number of rows and `siz(2)` is the number of columns.

`IND = sub2ind(siz,I1,I2,...,In)` returns the linear index equivalent to the `n` subscripts `I1,I2,...,In` for an array of size `siz`. `siz` is an `n`-element vector that specifies the size of each array dimension.

Examples

Create a 3-by-4-by-2 array, `A`.

```
A = [17 24 1 8; 2 22 7 14; 4 6 13 20];
A(:,:,2) = A - 10
```

```
A(:,:,1) =

    17    24     1     8
     2    22     7    14
     4     6    13    20


A(:,:,2) =

     7    14    -9    -2
    -8    12    -3     4
    -6    -4     3    10
```

The value at row 2, column 1, page 2 of the array is -8.

```
A(2,1,2)

ans =

    -8
```

To convert `A(2,1,2)` into its equivalent single subscript, use `sub2ind`.

```
sub2ind(size(A),2,1,2)

ans =

    14
```

You can now access the same location in `A` using the single subscripting method.

```
A(14)

ans =

    -8
```

Backpropagation

```
% One-hot encoding
%
d = zeros(10, 1);
d(sub2ind(size(d), D(k), 1)) = 1;

% Backpropagation
%
e      = d - y;                    % Output layer
delta  = e;
```

```
    e5     = Wo' * delta;            % Hidden(ReLU) layer
    delta5 = (y5 > 0) .* e5;

    e4     = W5' * delta5;           % Pooling layer

    e3     = reshape(e4, size(y3));

    e2 = zeros(size(y2));
    W3 = ones(size(y2)) / (2*2);
    for c = 1:20
      e2(:, :, c) = kron(e3(:, :, c), ones([2 2])) .* W3(:, :, c);
    end

    delta2 = (y2 > 0) .* e2;         % ReLU layer

    delta1_x = zeros(size(W1));      % Convolutional layer
    for c = 1:20
      delta1_x(:, :, c) = conv2(x(:, :), rot90(delta2(:, :, c), 2),
'valid');
    end

    dW1 = dW1 + delta1_x;
    dW5 = dW5 + delta5*y4';
    dWo = dWo + delta *y5';
  end

  % Update weights
  %
  dW1 = dW1 / bsize;
  dW5 = dW5 / bsize;
  dWo = dWo / bsize;

  momentum1 = alpha*dW1 + beta*momentum1;
  W1        = W1 + momentum1;

  momentum5 = alpha*dW5 + beta*momentum5;
  W5        = W5 + momentum5;

  momentumo = alpha*dWo + beta*momentumo;
  Wo        = Wo + momentumo;
end
```

Backprogation of Errors

```
...
e      = d - y;
delta  = e;

e5     = Wo' * delta;
delta5 = e5 .* (y5> 0);

e4     = W5' * delta5;
e3     = reshape(e4, size(y3));
...
```

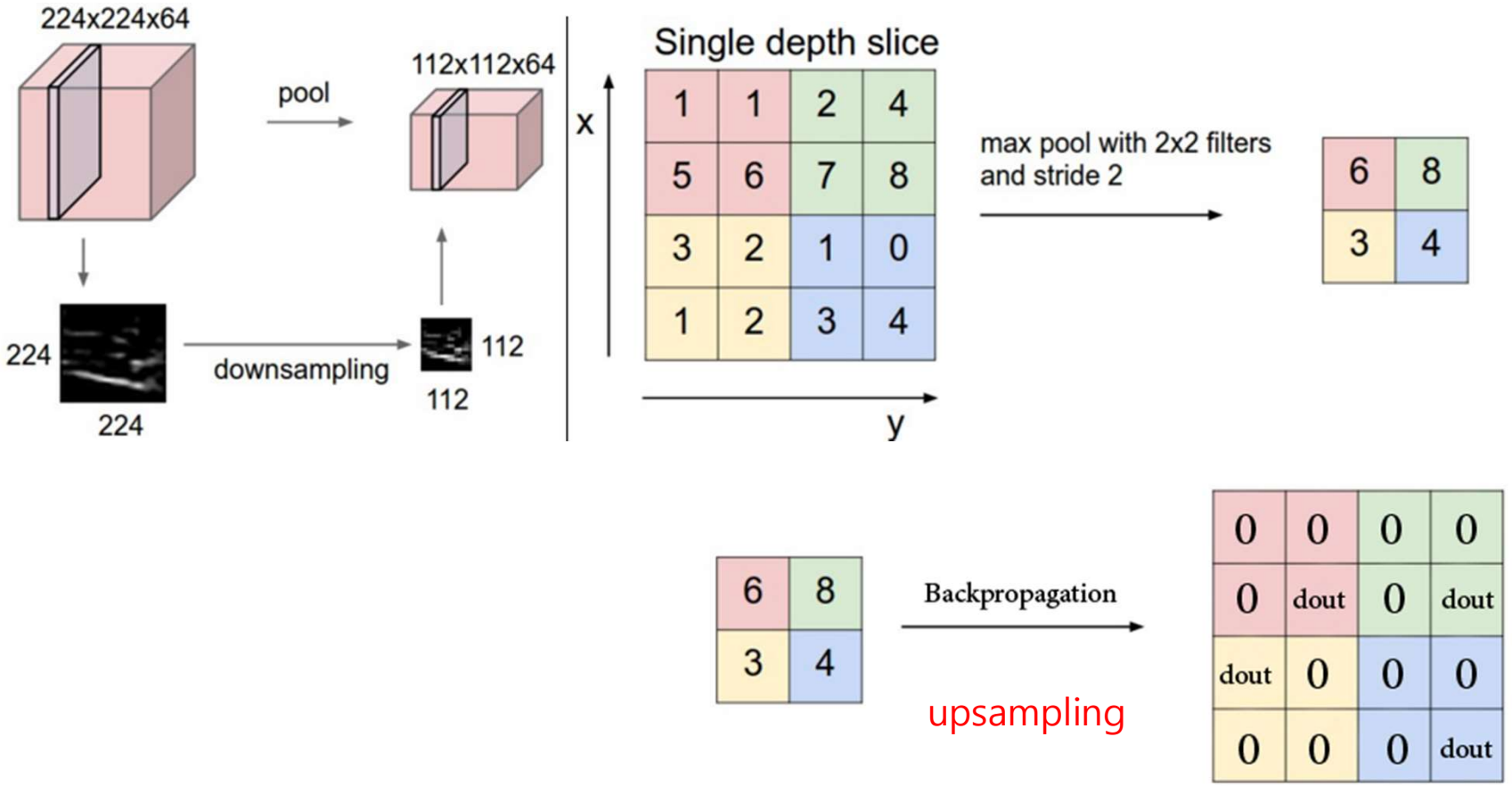
Pooling and Convolution Layers in Backpropagation

```
...
e2 = zeros(size(y2));          % Pooling
W3 = ones(size(y2)) / (2*2);
for c = 1:20
  e2(:, :, c) = kron(e3(:, :, c), ones([2 2])) .* W3(:, :, c);
end

delta2 = (y2 > 0) .* e2;

delta1_x = zeros(size(W1));
for c = 1:20
  delta1_x(:, :, c) = conv2(x(:, :), rot90(delta2(:, :, c), 2),
'valid');
end
...
```

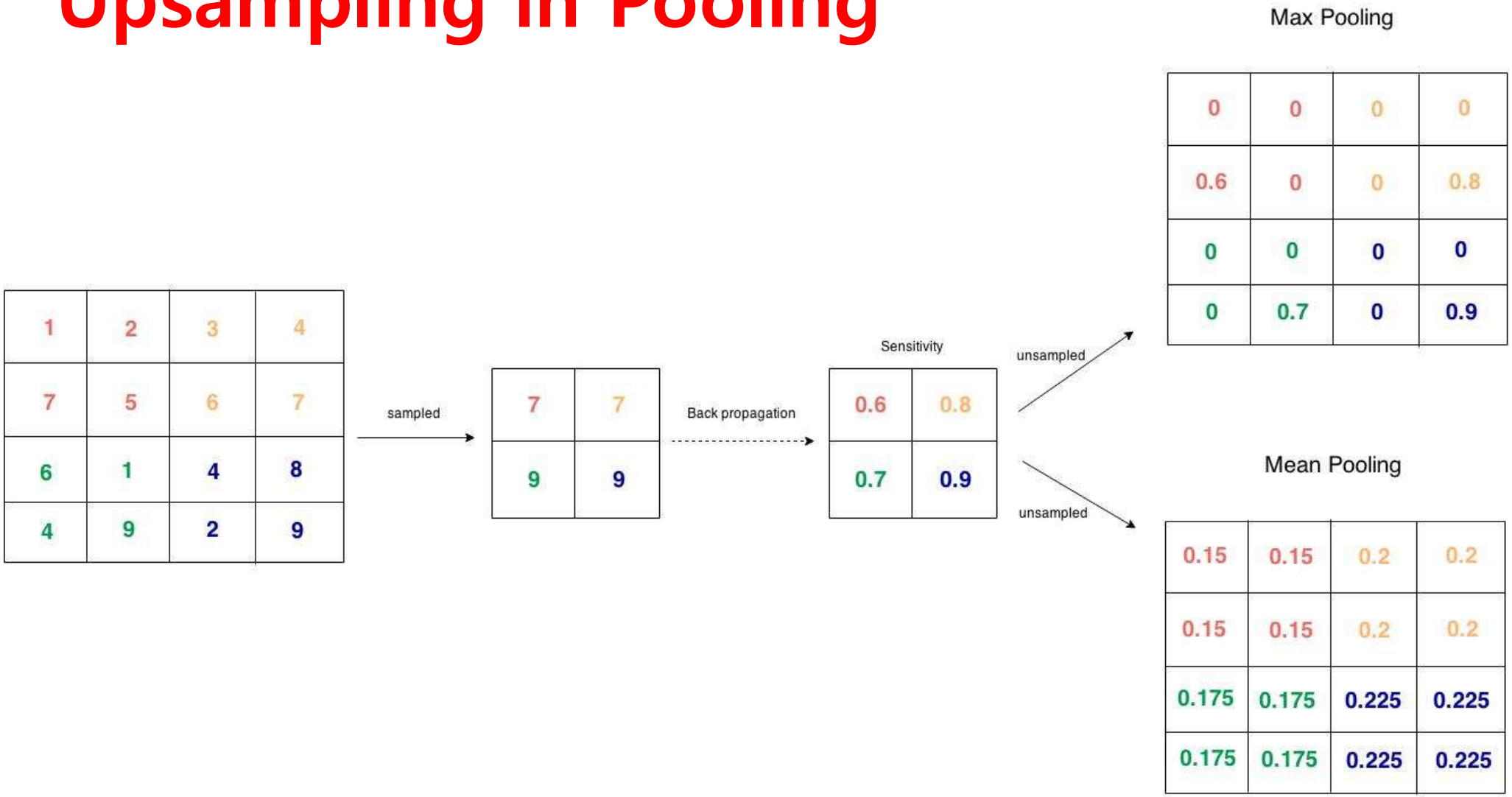
Note

Note

Forward and Backward Max Pooling



Upsampling in Pooling



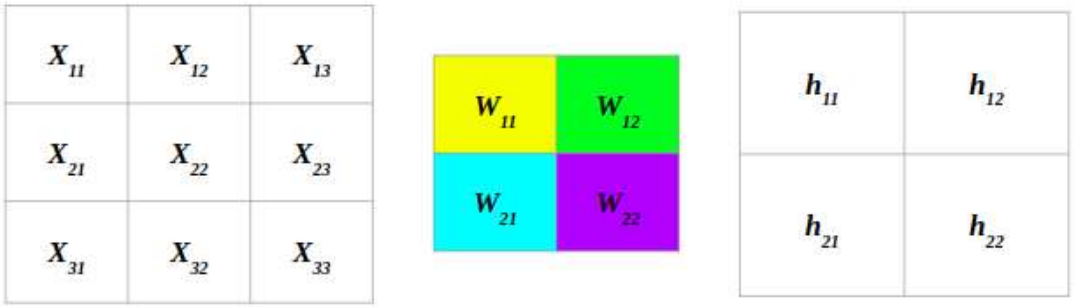
Convolution Operation (Forward Pass)

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Convolution Operation (Forward Pass)



Input Size : 3x3, **Filter Size** : 2x2, **Output Size** : 2x2

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

$$h_{22} = W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$$

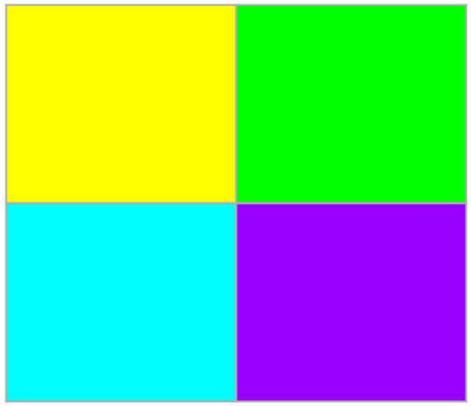
Output Equations

https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Derivative Computation (Backward Pass)

Original Image

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



∂h_{11}	∂h_{12}
∂h_{21}	∂h_{22}

Derivatives of Convolved Image

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$

$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$

$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$

$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

Final derivatives of weights after performing backpropagation

Note: no deconvolution, we are trying to compute updates of weights

https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c

Full Derivation, Here

- **Only Numpy: Understanding Back Propagation for Transpose Convolution in Multi Layer CNN with Example and Interactive Code.**
- https://towardsdatascience.com/only-numpy-understanding-back-propagation-for-transpose-convolution-in-multi-layer-cnn-with-c0a07d191981

Finally Weight Updates

```
...
momentum1 = alpha*dW1 + beta*momentum1;
W1        = W1 + momentum1;

momentum5 = alpha*dW5 + beta*momentum5;
W5        = W5 + momentum5;

momentumo = alpha*dWo + beta*momentumo;
Wo        = Wo + momentumo;
...
```

Function Conv.m

```
function y = Conv(x, W)
%
%

[wrow, wcol, numFilters] = size(W);
[xrow, xcol, ~         ] = size(x);

yrow = xrow - wrow + 1;
ycol = xcol - wcol + 1;

y = zeros(yrow, ycol, numFilters);

for k = 1:numFilters
  filter = W(:, :, k);
  filter = rot90(squeeze(filter), 2);
  y(:, :, k) = conv2(x, filter, 'valid');
end

end
```

More details:
http://deeplearning.stanford.edu/tutorial/supervised/ExerciseConvolutionAndPooling/

Function Pool.m

```
function y = Pool(x)
%
% 2x2 mean pooling
%
[xrow, xcol, numFilters] = size(x);

y = zeros(xrow/2, xcol/2, numFilters);
for k = 1:numFilters
  filter = ones(2) / (2*2);    % for mean
  image  = conv2(x(:, :, k), filter, 'valid');

  y(:, :, k) = image(1:2:end, 1:2:end);
end

end
```

PlotFeatures.m

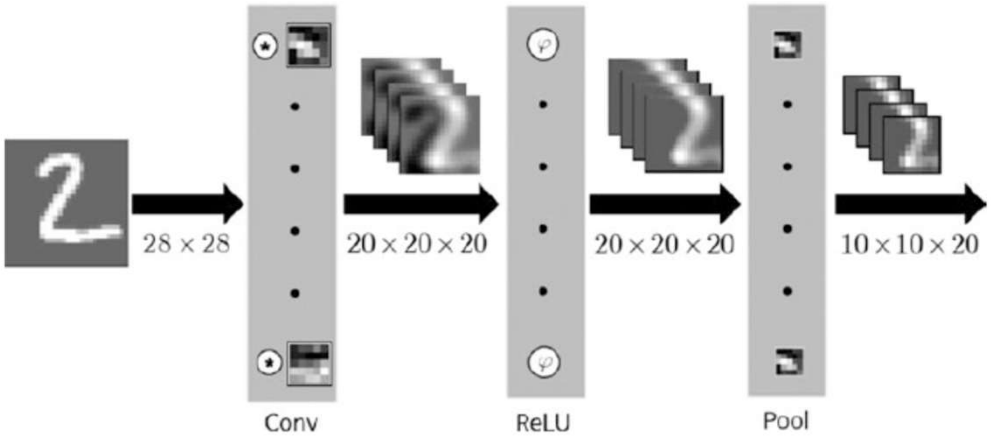


Figure 6-19. *How the image is processed while it passes through the convolution and pooling layers*