Convolution Neural Networks: [Part 2] and Implementation

Principles of Modeling for Cyber-Physical Systems
Module 3: Automotive CPS

Madhur Behl | University of Virginia

Slides Credits:
Fei-Fei Li
Andrej Karpathy
Justin Johnson

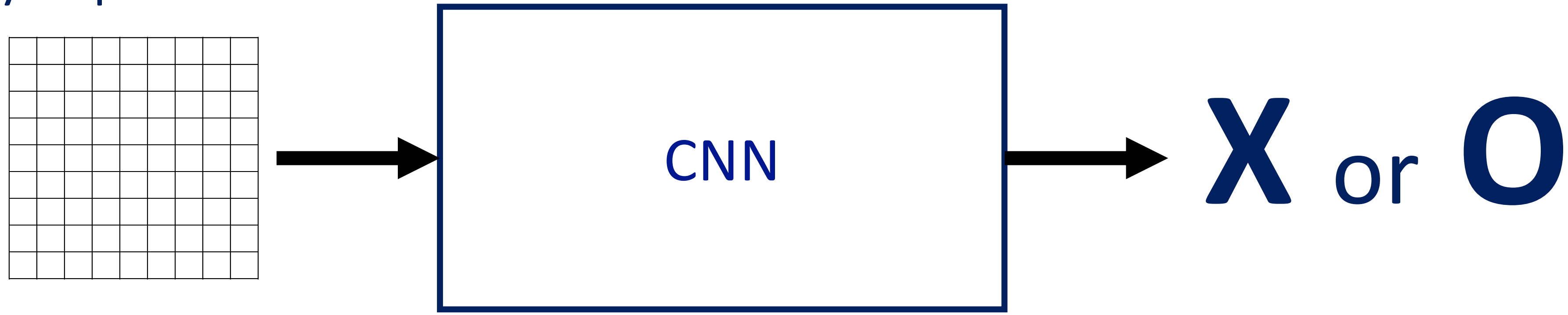
Announcement

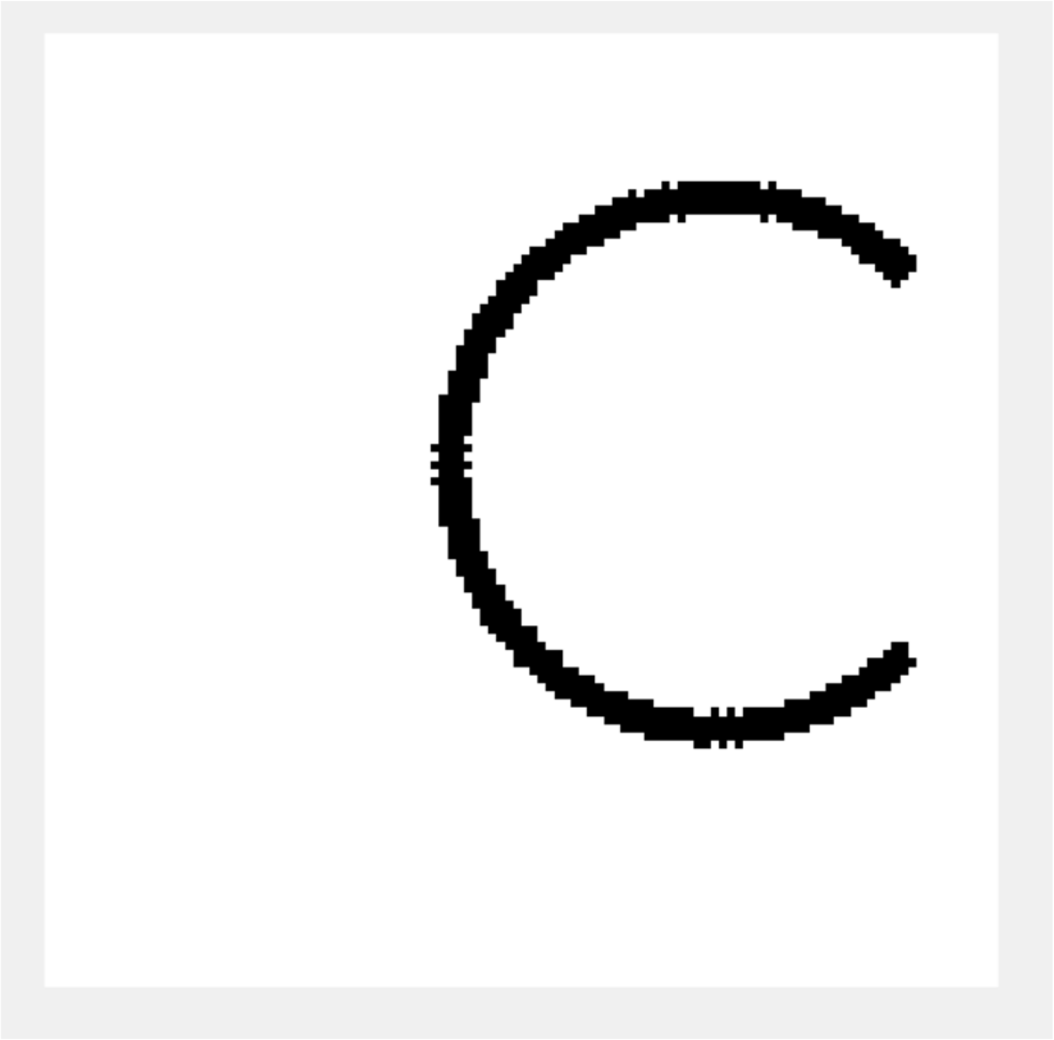
- Assignment 8 is out. Due in 1.5 weeks – Dec, 5, 2019.
- Train a CNN to categorize images as X or O.
- Template code in Matlab provided.
    - Not mandatory to use Matlab (more on this later).

Assignment 8 ConvNet: X’s and O’s

Says whether a picture is of an X or an O

A two-dimensional array of pixels





```
K>> size(example_image)

ans =

   116   116
```

What is provided:

1. Dataset of 900 images each of two categories
2. Template code for training and evaluating a CNN in MATLAB

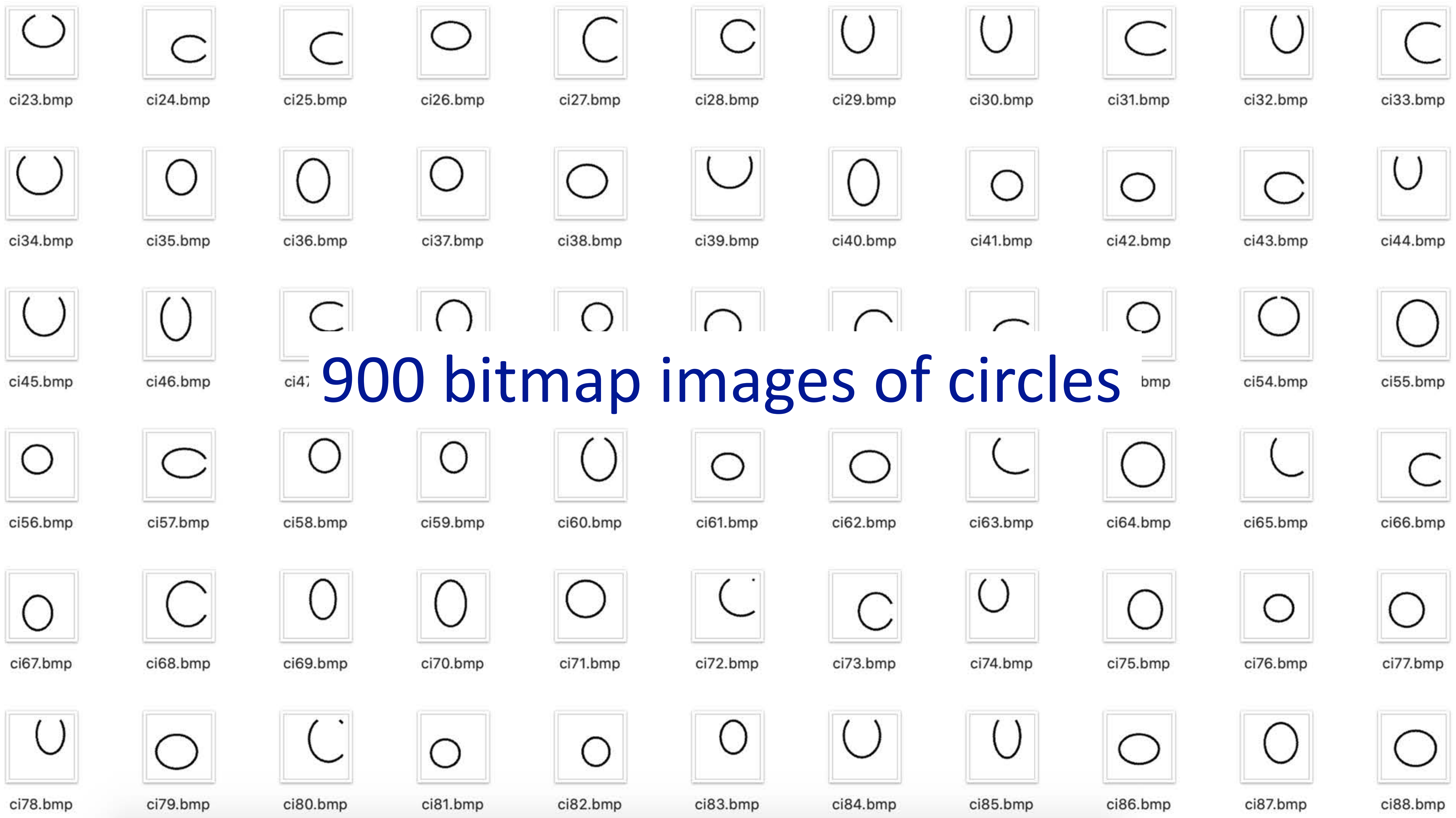
BasicCNNtemplate.m

training_data

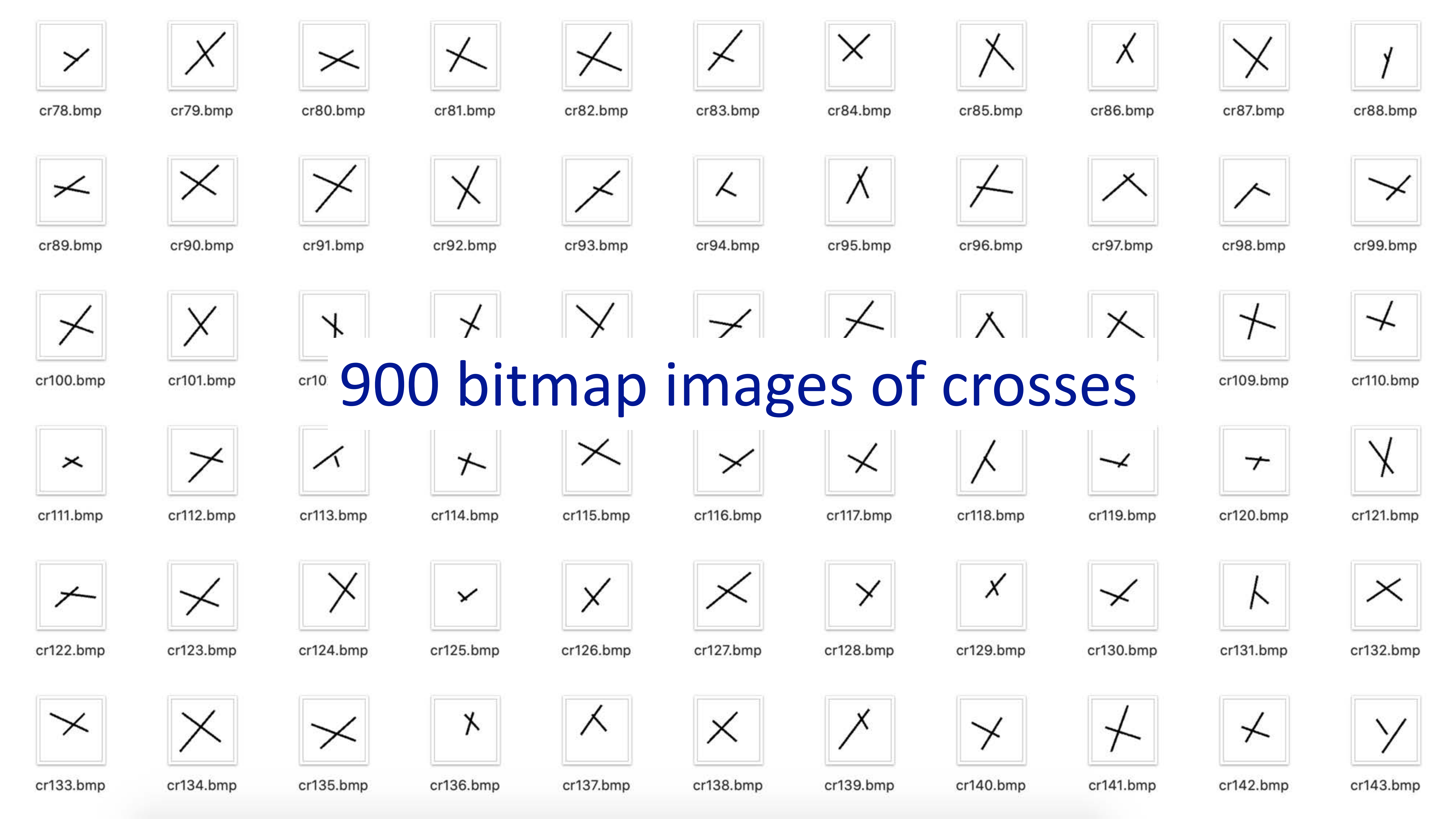
Root folder must contain the
training_data folder
for the template to work.

training_data contains two subfolders
- circles and crosses

900 bitmap images of circles



900 bitmap images of crosses



BasicCNNTemplate.m overview

1. Configure the execution of the code.
2. Load and prep the data
3. Setup the CNN architecture
4. Train the Network
5. Test the performance of the CNN
6. Plotting code.

1. Configure the execution of the code.

```
doTraining                = true;

% Set these flags to inspect and plot the network (Note: optimized for screen resolution (1920x1200))
show.wrong_classified     = false;                           % wrong classified images
show.filter               = false;                           % filters(weights)
show.feature_maps         = true;                            % feature maps
```

2. Load and prep the data

Create an image datastore object

```
IMDS = imageDatastore('training_data','IncludeSubfolders',true,'FileExtensions','.bmp','LabelSource','foldernames');

example_image = readimage(IMDS,1);   % read one example image from the datastore.

% Uncomment the line below to display the example_image.
% imshow(example_image);
```

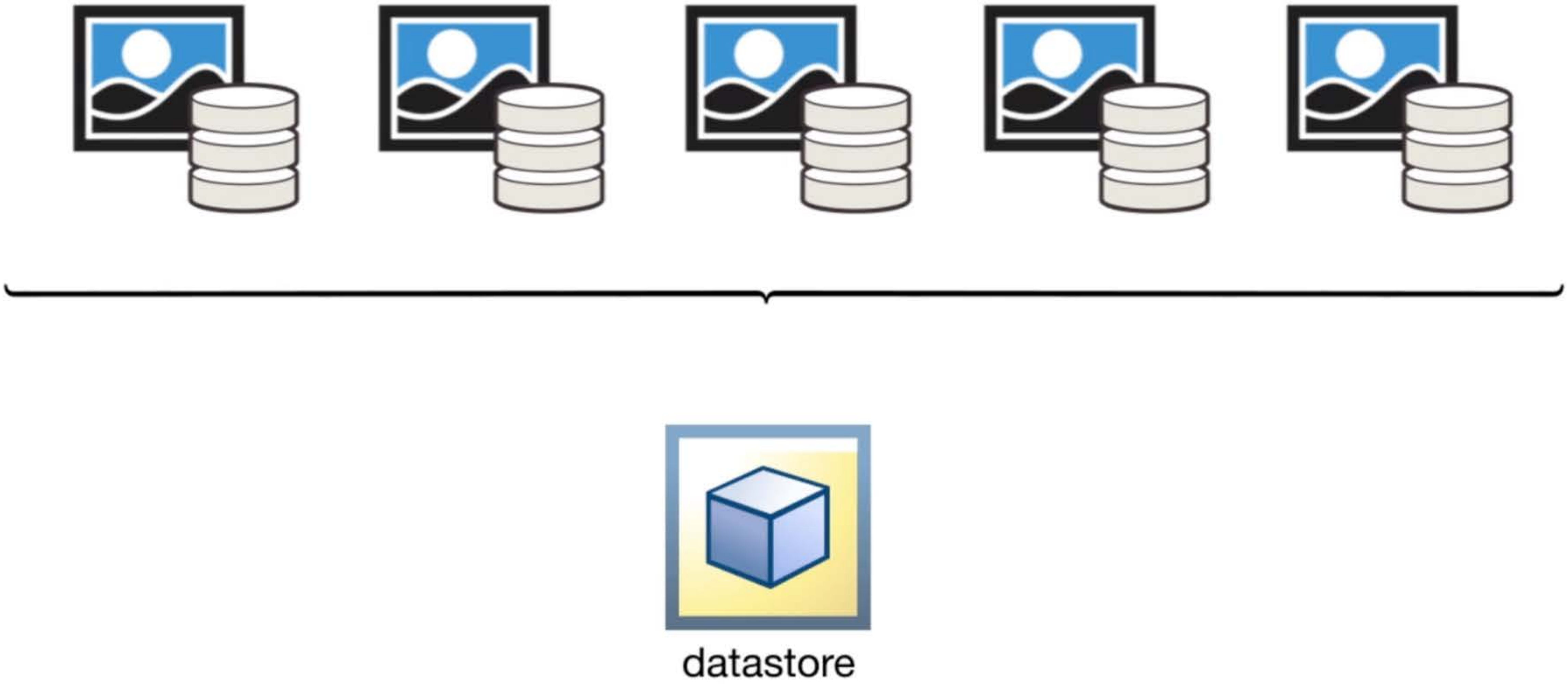
Get channel info and # of label categories

```
numChannels = size(example_image,3);                % get color information - The images are single channel in th
numImageCategories = size(categories(IMDS.Labels),1);  % Two image categories in our dataset.

% Create the training and testing datasets.
% Split ImageDatastore labels by proportions
training_propotion = 0.7;
[trainingDS,validationDS] = splitEachLabel(IMDS,training_propotion,'randomize');
```

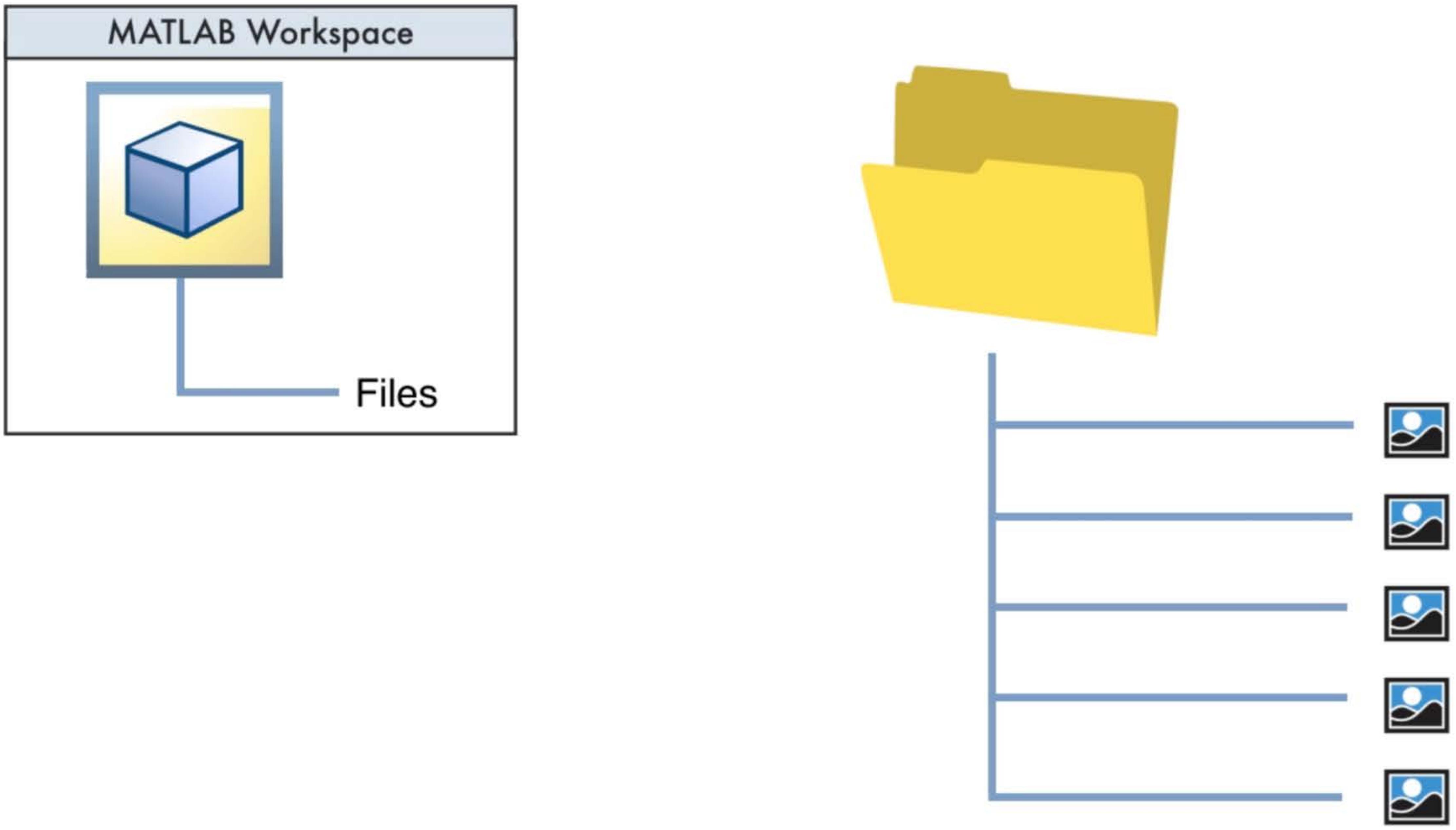
Partition data into training and validation

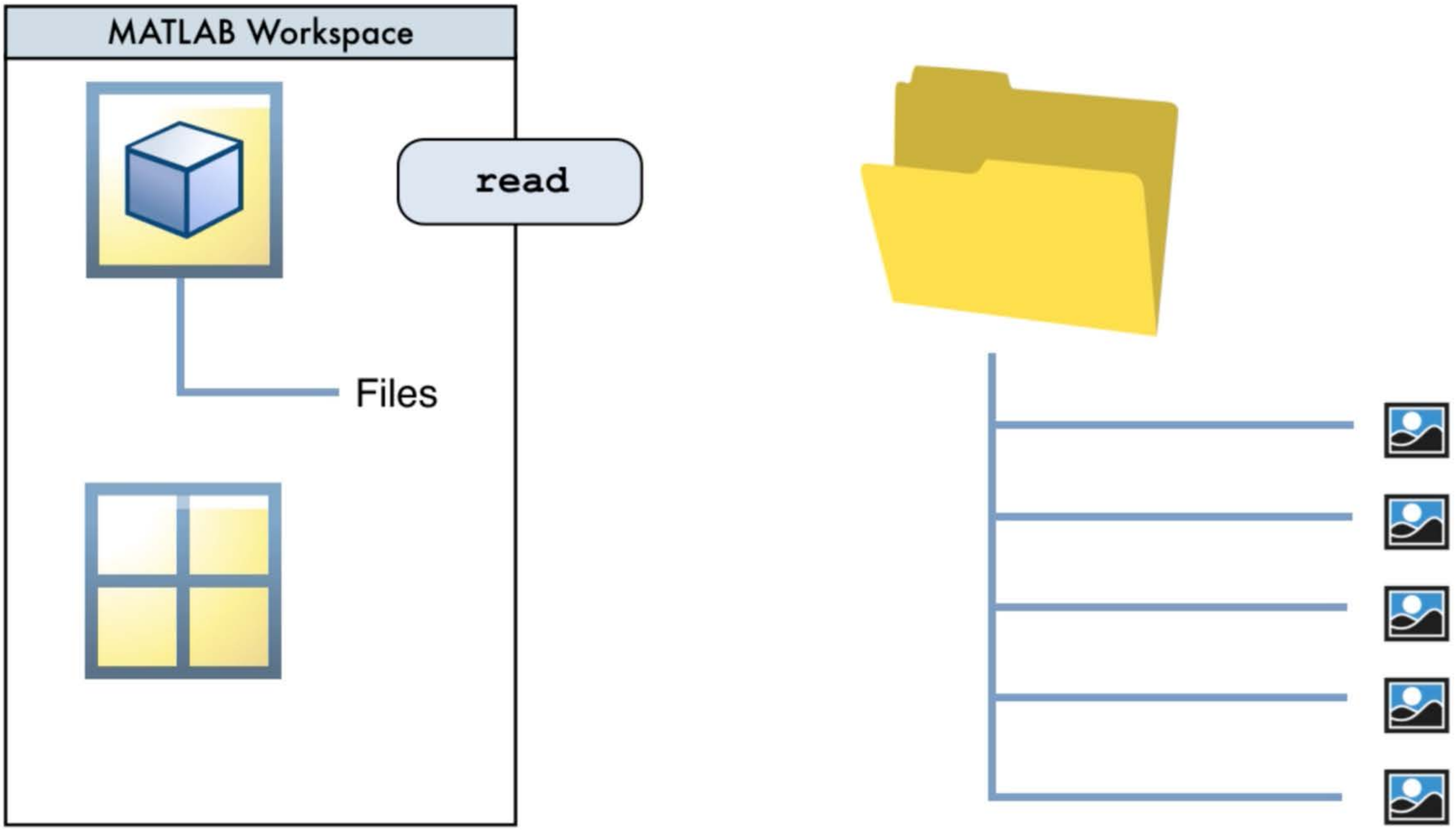
```
LabelCntTr = countEachLabel(trainingDS);                % load lable information
LabelCntVa = countEachLabel(validationDS);
```

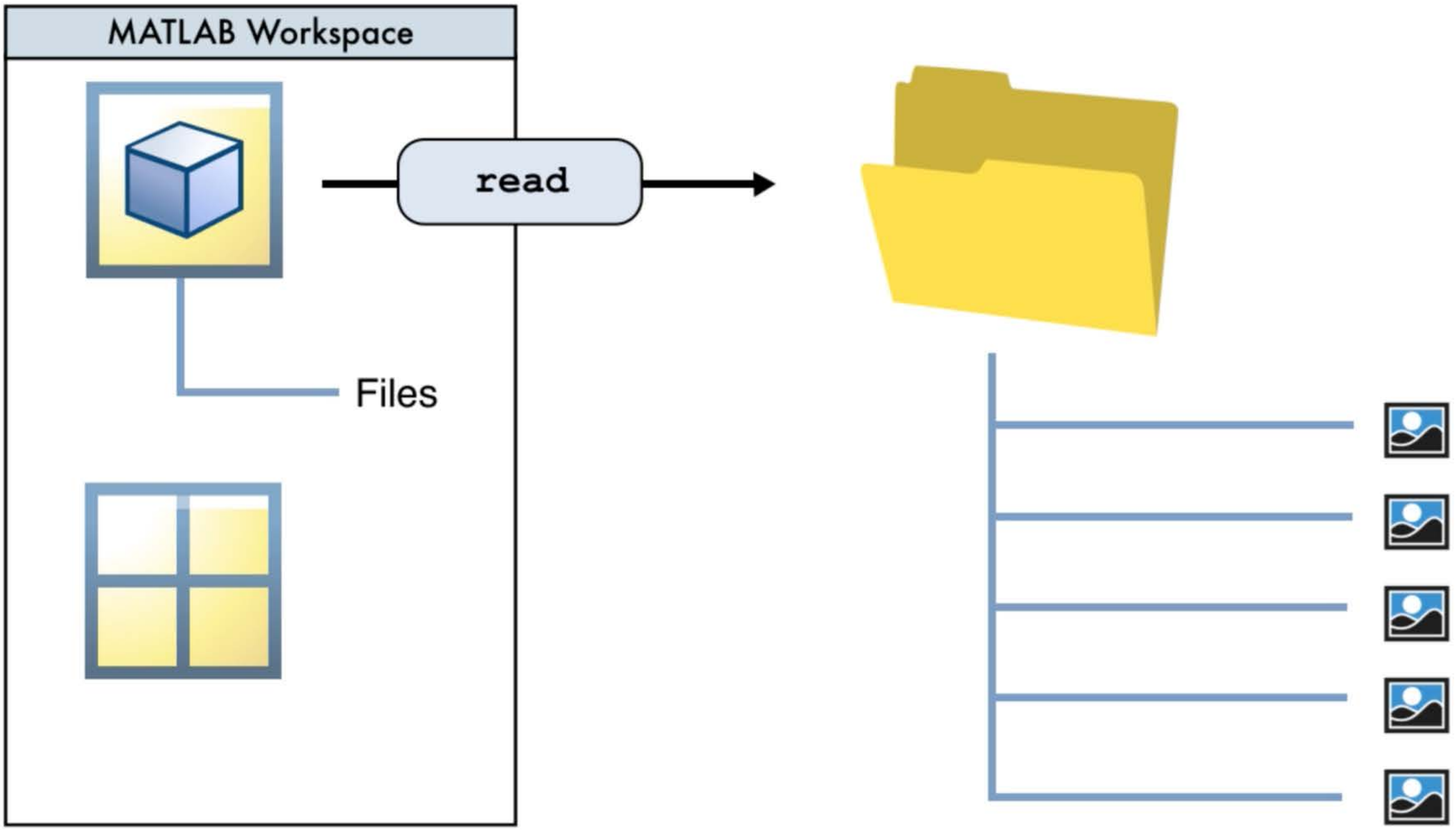
630 samples in training, 270 in validation for proportion =0.7


datastore


Files


MATLAB Workspace
Files

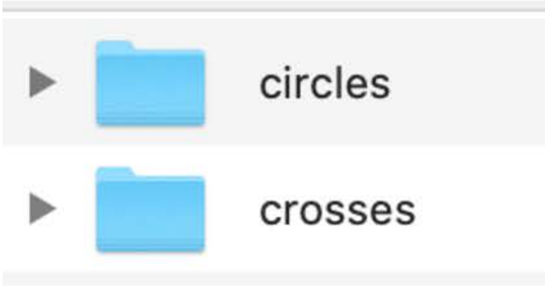

MATLAB Workspace
read
Files


MATLAB Workspace
read
Files

BasicCNNtemplate.m

training_data

Root folder must contain the training_data folder
for the template to work.



training_data contains two subfolders
- circles and crosses

2. Load and prep the data

Create an image datastore object

```
IMDS = imageDatastore('training_data','IncludeSubfolders',true,'FileExtensions','.bmp','LabelSource','foldernames');

example_image = readimage(IMDS,1);    % read one example image from the datastore.

% Uncomment the line below to display the example_image.
% imshow(example_image);
```

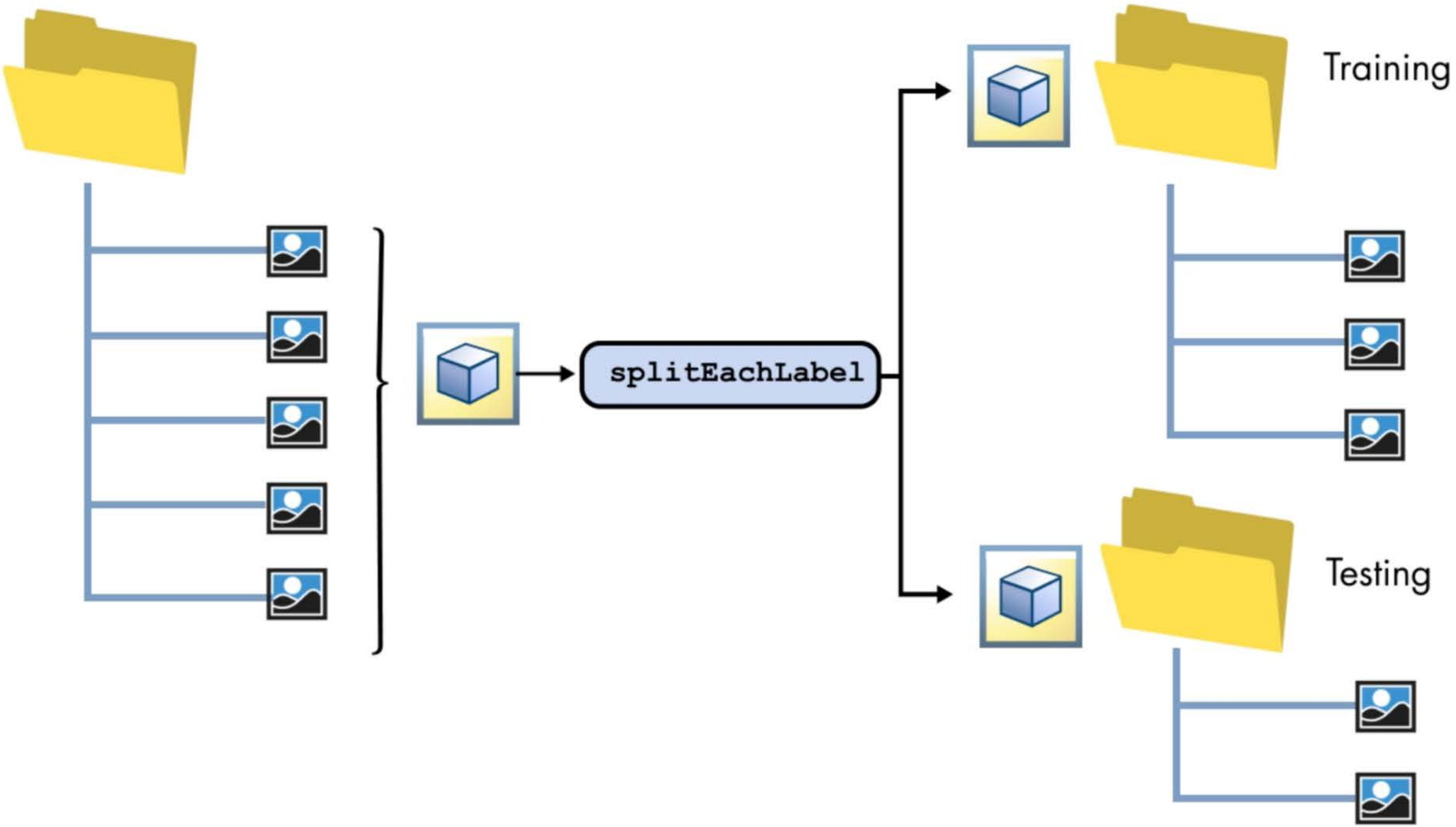
Get channel info and # of label categories

```
numChannels = size(example_image,3);                    % get color information - The images are single channel in th
numImageCategories = size(categories(IMDS.Labels),1);   % Two image categories in our dataset.

% Create the training and testing datasets.
% Split ImageDatastore labels by proportions
```

Partition data into training and validation

```
training_propotion = 0.7;
[trainingDS,validationDS] = splitEachLabel(IMDS,training_propotion,'randomize');

LabelCntTr = countEachLabel(trainingDS);                % load lable information
LabelCntVa = countEachLabel(validationDS);
```

630 samples in training, 270 in validation for proportion =0.7



Training
splitEachLabel
Testing

3. Setup the CNN architecture

```
%% Setup of the CNN architecture.

if doTraining

    % Convolutional layer parameters
    filterSize = [10 10];
    numFilters = 16;

    % An image input layer inputs 2-D images to a network and applies data normalization.
    % The size of the layer is the same as the number of pixels in our
    % input images.
    inputLayer = imageInputLayer(size(example_image),'Name','Input');  % no data augmentation
```

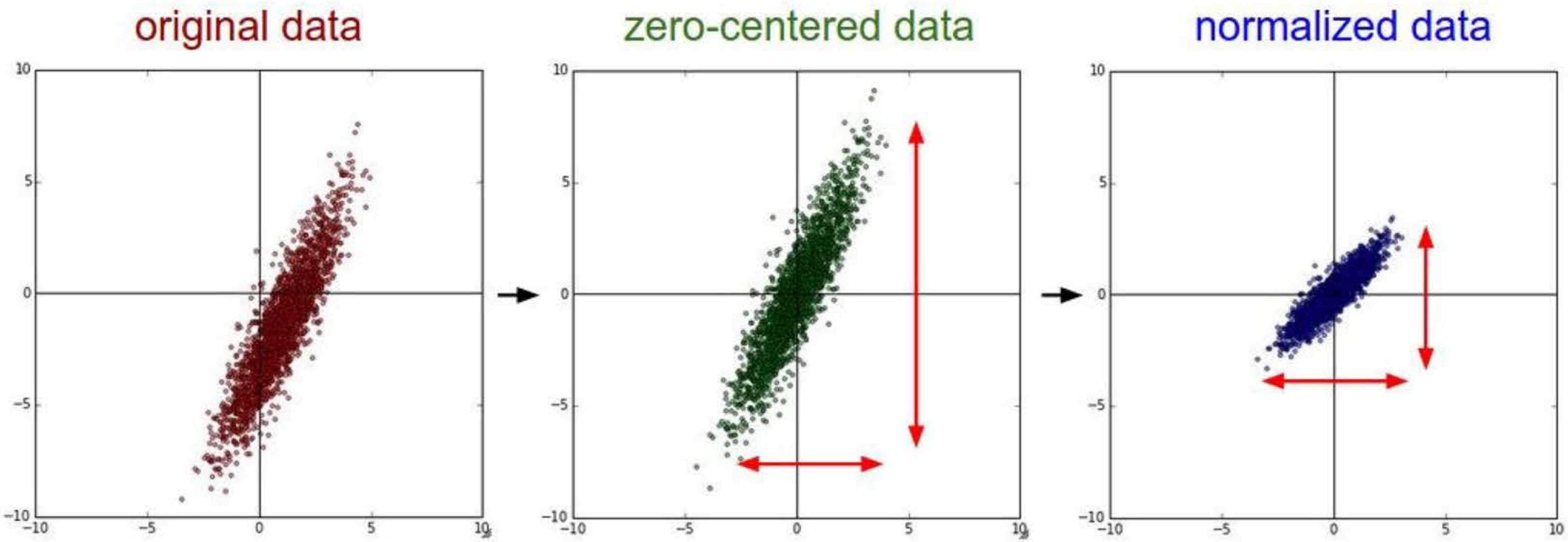
You can change the filter size or even try multiple filter sizes

Number of filters usually a power of 2

Create the input layer which simply reads the 116x116 bmp image.

Note the use of the ‘Name’, ‘layer name’ args

Data Preprocessing




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

Image Data Preprocessing

Image Data Preprocessing

Crop to Symmetric Aspect Ratio

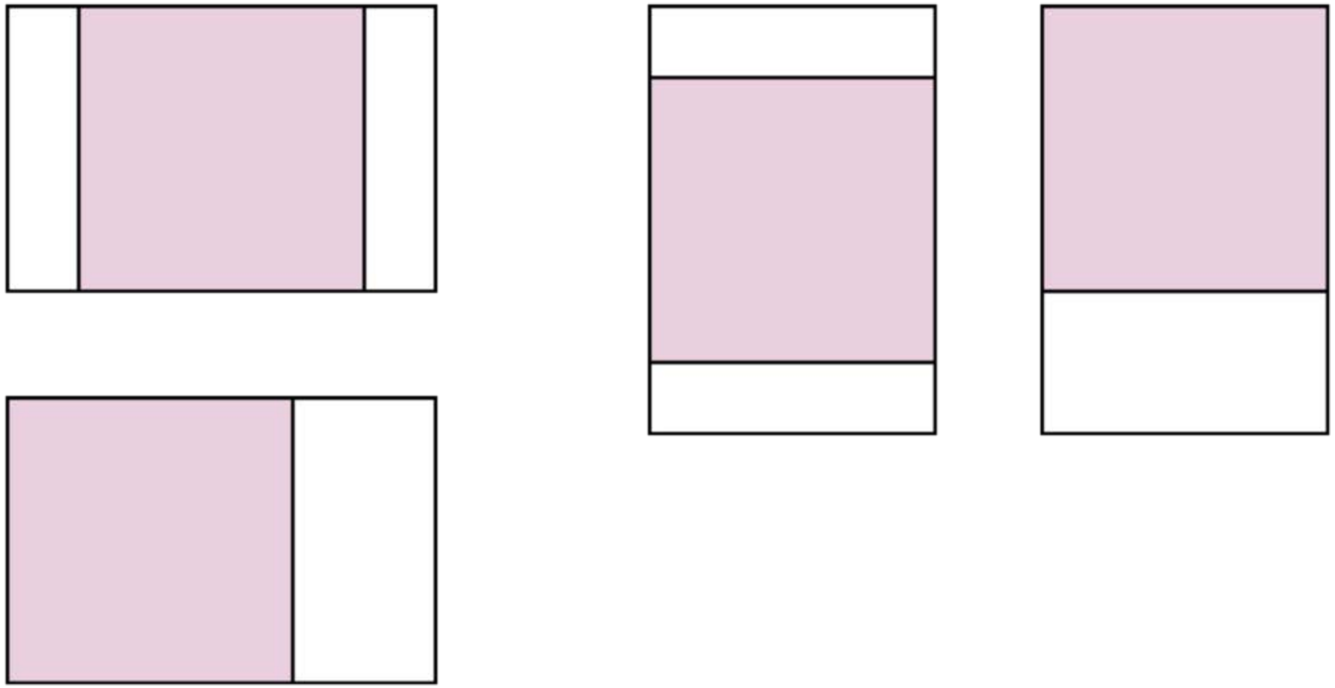


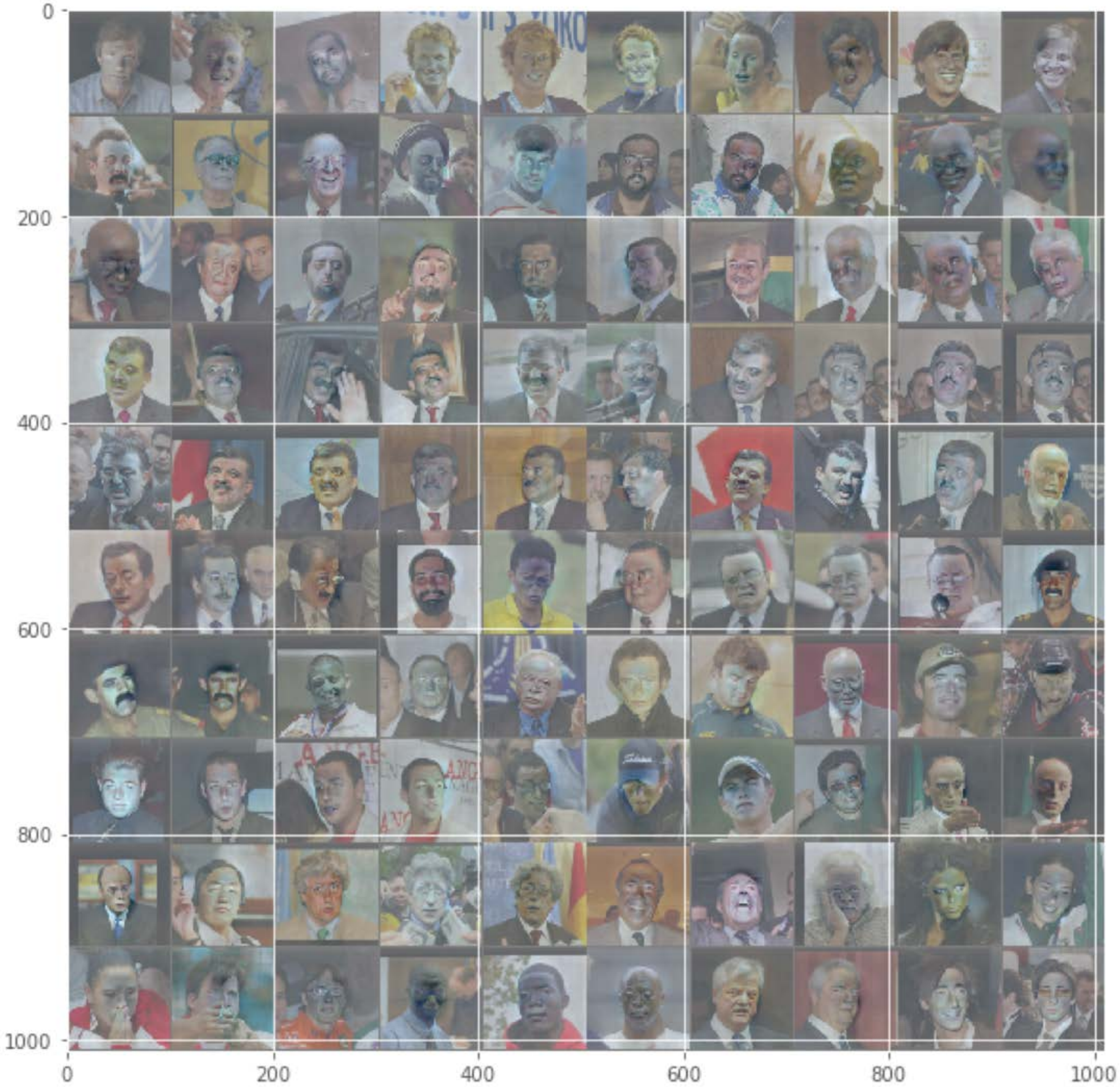
Image Data Preprocessing

Pixel wise mean and std deviation

Image Data Preprocessing

Zero Center Normalization

- Subtract mean
- Divide by std dev



3. Setup the CNN architecture

You need to specify the layers in the architecture

```
middleLayers = [
    % The first convolutional layer has a bank of numFilters filters of size filterSize.
    % A symmetric padding of 4 pixels is added.
    convolution2dLayer(...)
    % Next add the ReLU layer:
    reluLayer('Name','ReLu1')
   % Follow it with a max pooling layer that has a 5x5 spatial pooling area
    % and a stride of 2 pixels. This down-samples the data dimensions.
    maxPooling2dLayer(...)

    % Repeat the 3 core layers to complete the middle of the network.
    % This time use 32 filters instead of 16.


    % Repeat the 3 core layers one more time
    % This time change symmetric padding to 2 for the convolution, and
    % the stride to 3 for the maxpoolinglayer.


    ];
```

3. Setup the CNN architecture

Example architecture

1	'Input'	Image Input	116x116x1 images with 'zerocenter' normalization
2	'Conv1'	Convolution	16 10x10x1 convolutions with stride [1 1] and padding [4 4 4 4]
3	'ReLu1'	ReLU	ReLU
4	'Pool1'	Max Pooling	5x5 max pooling with stride [2 2] and padding [0 0 0 0]
5	'Conv2'	Convolution	32 10x10 convolutions with stride [1 1] and padding [4 4 4 4]
6	'ReLu2'	ReLU	ReLU
7	'Pool2'	Max Pooling	5x5 max pooling with stride [2 2] and padding [0 0 0 0]
8	'Conv3'	Convolution	32 10x10 convolutions with stride [1 1] and padding [2 2 2 2]
9	'ReLu3'	ReLU	ReLU
10	'Pool3'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
11	'FC'	Fully Connected	2 fully connected layer
12	'Softmax'	Softmax	softmax
13	'Classification'	Classification Output	crossentropyex

3. Setup the CNN architecture – Useful functions

Convolution and Fully Connected Layers

Layer	Description
convolution2dLayer	A 2-D convolutional layer applies sliding convolutional filters to the input.
convolution3dLayer	A 3-D convolutional layer applies sliding cuboidal convolution filters to three-dimensional input.
groupedConvolution2dLayer	A 2-D grouped convolutional layer separates the input channels into groups and applies sliding convolutional filters. Use grouped convolutional layers for channel-wise separable (also known as depth-wise separable) convolution.
transposedConv2dLayer	A transposed 2-D convolution layer upsamples feature maps.
transposedConv3dLayer	A transposed 3-D convolution layer upsamples three-dimensional feature maps.
fullyConnectedLayer	A fully connected layer multiplies the input by a weight matrix and then adds a bias vector.

3. Setup the CNN architecture – Useful functions

Activation Layers

Layer	Description
reluLayer	A ReLU layer performs a threshold operation to each element of the input, where any value less than zero is set to zero.
leakyReluLayer	A leaky ReLU layer performs a threshold operation, where any input value less than zero is multiplied by a fixed scalar.
clippedReluLayer	A clipped ReLU layer performs a threshold operation, where any input value less than zero is set to zero and any value above the *clipping ceiling* is set to that clipping ceiling.
eluLayer	An ELU activation layer performs the identity operation on positive inputs and an exponential nonlinearity on negative inputs.
tanhLayer	A hyperbolic tangent (tanh) activation layer applies the tanh function on the layer inputs.
preluLayer (Custom layer example)	A PReLU layer performs a threshold operation, where for each channel, any input value less than zero is multiplied by a scalar learned at training time.

3. Setup the CNN architecture – Useful functions

Pooling and Unpooling Layers

Layer	Description
averagePooling2dLayer	An average pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the average values of each region.
averagePooling3dLayer	A 3-D average pooling layer performs down-sampling by dividing three-dimensional input into cuboidal pooling regions and computing the average values of each region.
globalAveragePooling2dLayer	A global average pooling layer performs down-sampling by computing the mean of the height and width dimensions of the input.
globalAveragePooling3dLayer	A global average pooling layer performs down-sampling by computing the mean of the height, width, and depth dimensions of the input.
maxPooling2dLayer	A max pooling layer performs down-sampling by dividing the input into rectangular pooling regions, and computing the maximum of each region.
maxPooling3dLayer	A 3-D max pooling layer performs down-sampling by dividing three-dimensional input into cuboidal pooling regions, and computing the maximum of each region.
maxUnpooling2dLayer	A max unpooling layer unpools the output of a max pooling layer.

3. Setup the CNN architecture

Final layers already defined – need not change

```
finalLayers = [

    % % Add a fully connected layer with the same number of neurons as
    % the number of image categories.
    fullyConnectedLayer(numImageCategories,'Name','FC')

    % Add the softmax loss layer and classification layer.
    % The final layers use the output of the fully connected layer to compute the categorical
    % probability distribution over the image classes. During the training
    % process, all the network weights are tuned to minimize the loss over this
    % categorical distribution.
    softmaxLayer('Name','Softmax');
    classificationLayer('Name','Classification')
    ];

layers = [
    inputLayer
    middleLayers
    finalLayers
    ];
```

Fully connected layer

Softmax layer

Cross entropy classification loss

All layers are stacked together

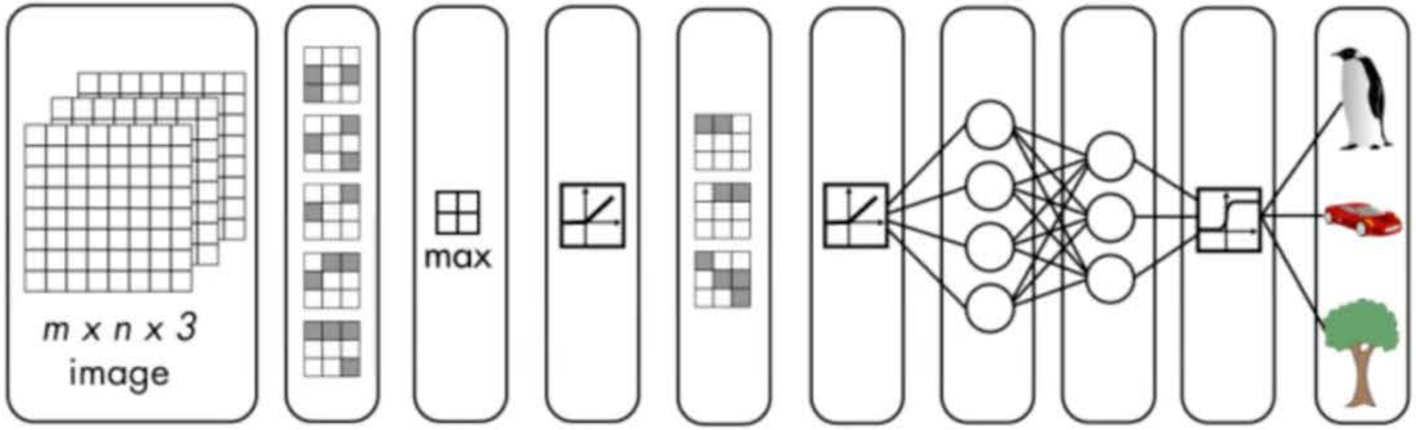
4. CNN Training

```
%% Train the Network
    %Initialize the first convolutional layer weights using
    % normally distributed random numbers with standard deviation of 0.0001.
    % This helps improve the convergence of training.
    layers(2).Weights = 0.0001 * randn([filterSize numChannels numFilters]);

    % Set the network training options
    % Try Momentum option 0.1 and 0.9 - Which is Better ?
    % Try LearningRate 0.01, and 0.001 - What is the difference ?
    % Try 10-20 Maxepochs

    opts = trainingOptions('sgdm', ...
        'Momentum', 0, ...
        'InitialLearnRate', 0, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.5, ...
        'LearnRateDropPeriod', 10, ...
        'L2Regularization', 0.004, ...
        'MaxEpochs', 0, ...
        'MiniBatchSize', 64, ...    % 64 for Quadro
        'Verbose', true,...
        'Plots','training-progress');

    % Train a network.
    rng('default');
    rng(123); % random seed

    XONet = trainNetwork(trainingDS, layers, opts);
    save('XONet.mat','XONet');
```

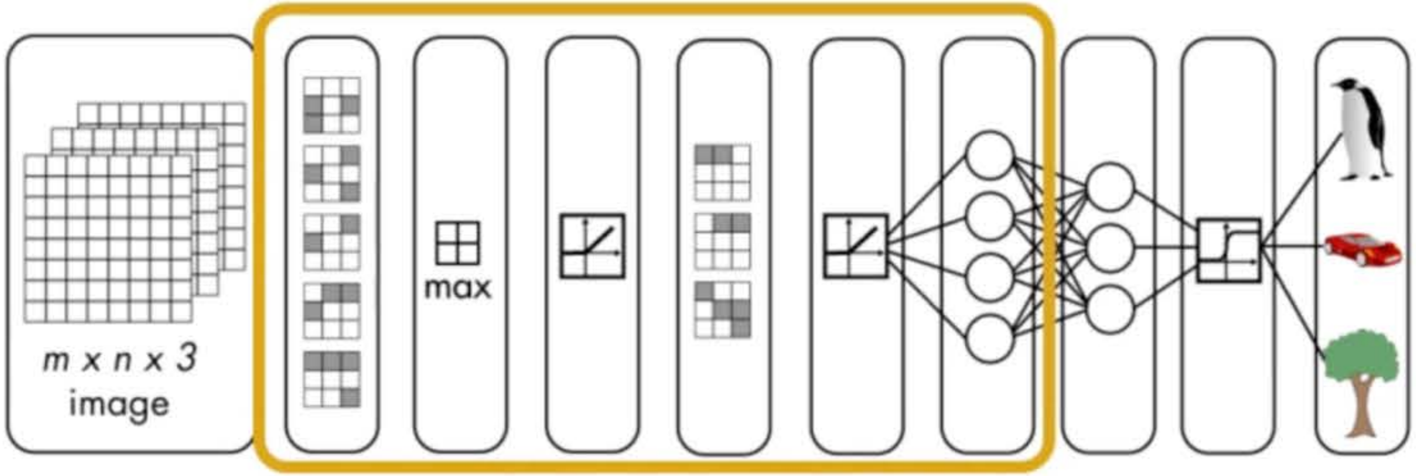
Initial weights have been provided

You have to try out different values for Momentum, Learning Rates and MaxEpochs

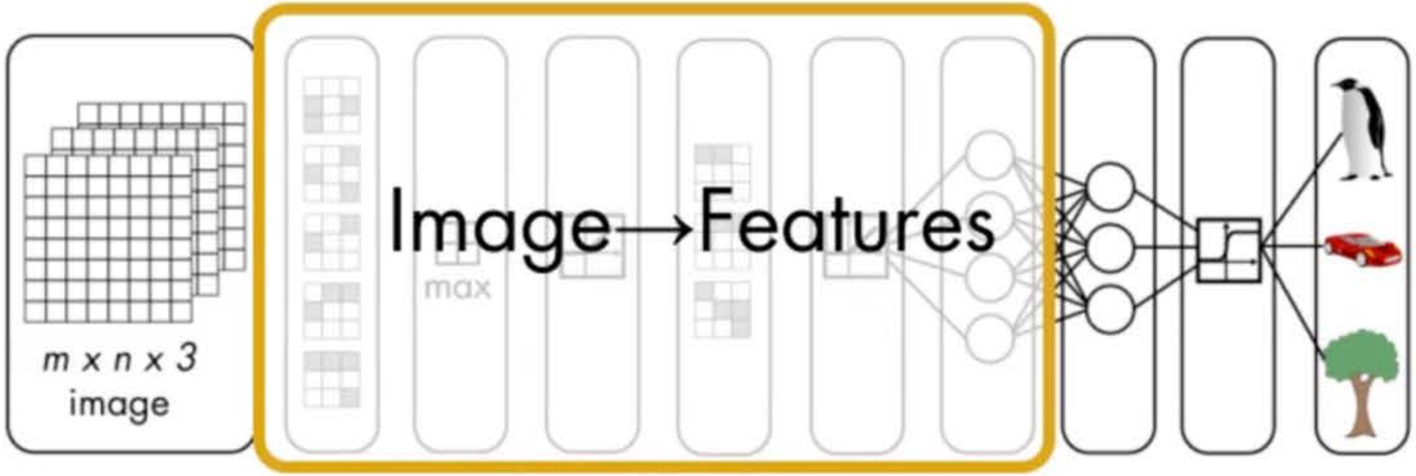
Training happens here
Should take ~ 10mins on a CPU



1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

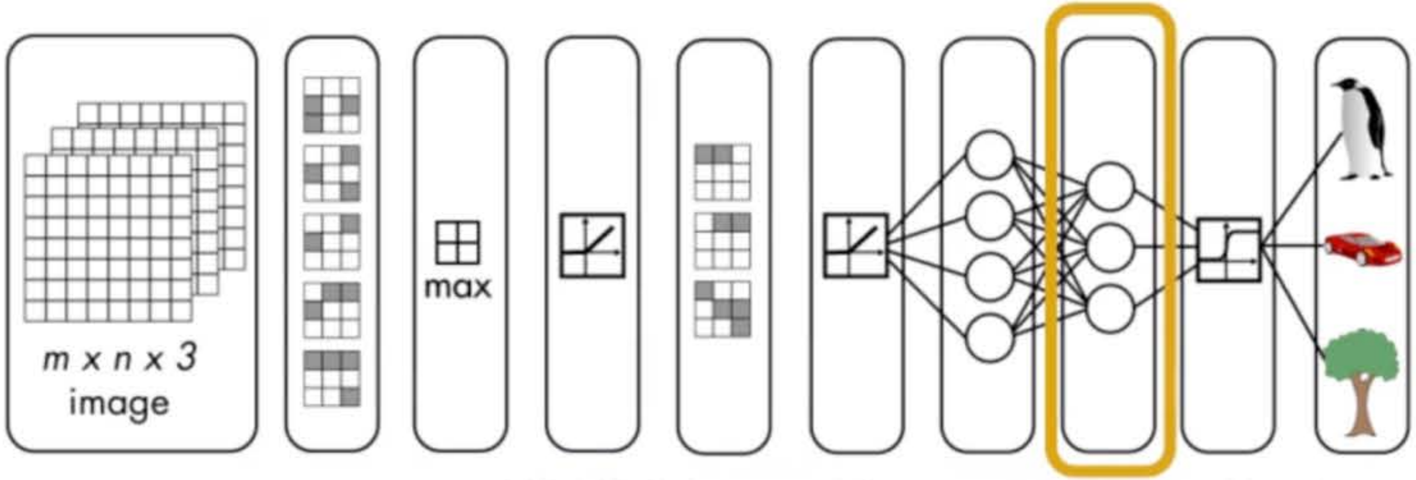


1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

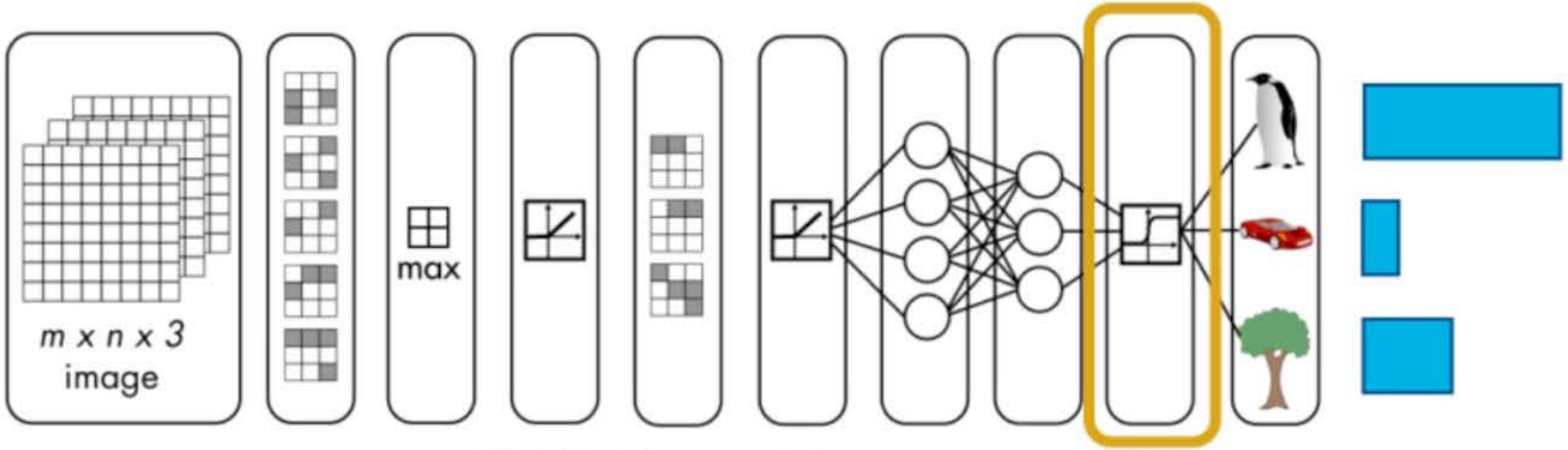


Image→Features

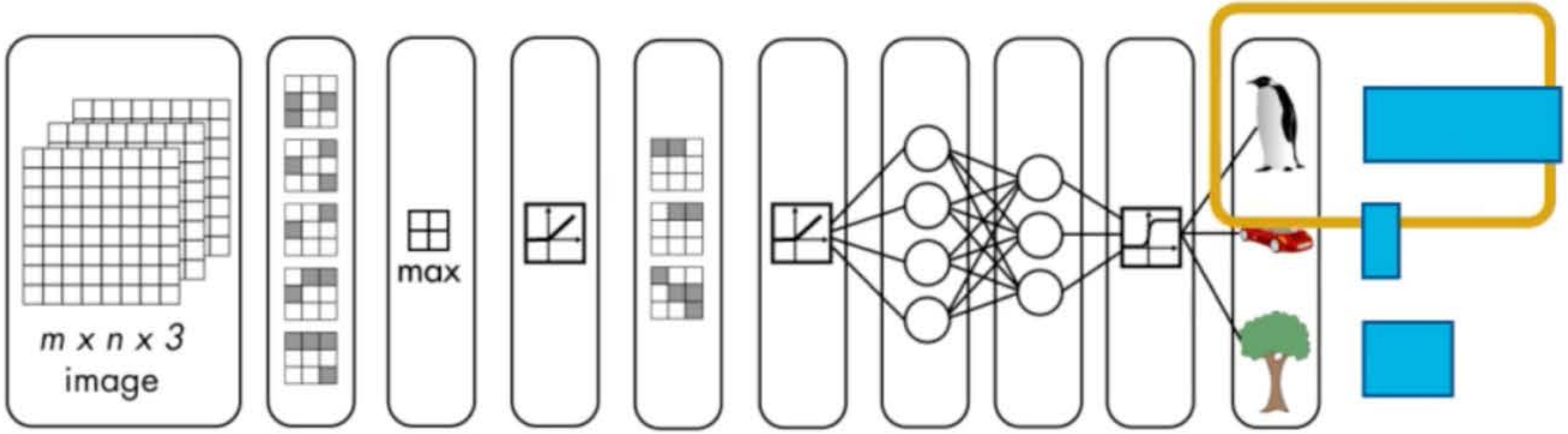
1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	[illegible] stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes



1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

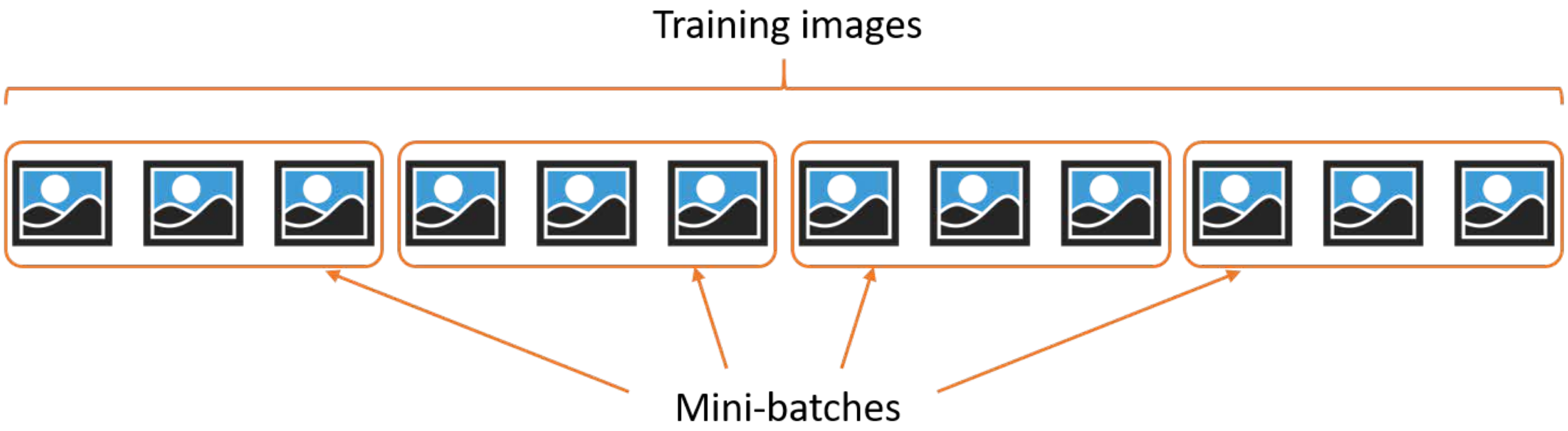


1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes



1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

```
Training on single GPU.
Initializing image normalization.
|========================================================================================|
|    Epoch      |    Iteration    |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning|
|               |                 |   (seconds)    |     Loss     |   Accuracy   |      Rate     |
|========================================================================================|
|             1 |               1 |           0.47 |       3.5061 |        7.81% |        0.0010 |
|             3 |              10 |          10.31 |       0.7686 |       75.00% |        0.0010 |
```



```
>> newnet = trainNetwork(net,data,options)


Training on single GPU.
Initializing image normalization.
|=======================================================================================================|
|     Epoch      |   Iteration    | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|                |                |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|=======================================================================================================|
|              1 |              1 |         0.47 |       3.5061 |        7.81% |       0.0010 |
|              3 |             10 |        10.31 |       0.7686 |       75.00% |       0.0010 |
|              5 |             20 |        18.96 |       0.2371 |       92.19% |       0.0010 |
|              8 |             30 |        27.43 |       0.0770 |       97.66% |       0.0010 |
|             10 |             40 |        35.31 |       0.0336 |       99.22% |       0.0010 |
|             13 |             50 |        43.17 |       0.0289 |       99.22% |       0.0010 |
|             15 |             60 |        50.15 |       0.0104 |      100.00% |       0.0010 |
|             18 |             70 |        56.84 |       0.0072 |      100.00% |       0.0010 |
|             20 |             80 |        63.00 |       0.0210 |       99.22% |       0.0010 |
|             23 |             90 |        69.37 |       0.0035 |      100.00% |       0.0010 |
|             25 |            100 |        74.85 |       0.0027 |      100.00% |       0.0010 |
|             28 |            110 |        81.19 |       0.0053 |      100.00% |       0.0010 |
```

```
>> newnet = trainNetwork(net,data,options)
```

```
Training on single GPU.
Initializing image normalization.
```

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.47	3.5061	7.81%	0.0010
3	10	10.31	0.7686	75.00%	0.0010
5	20	18.96	0.2371	92.19%	0.0010
8	30	27.43	0.0770	97.66%	0.0010
10	40	35.31	0.0336	99.22%	0.0010
13	50	43.17	0.0289	99.22%	0.0010
15	60	50.15	0.0104	100.00%	0.0010
18	70	56.84	0.0072	100.00%	0.0010
20	80	63.00	0.0210	99.22%	0.0010
23	90	69.37	0.0035	100.00%	0.0010
25	100	74.85	0.0027	100.00%	0.0010
28	110	81.19	0.0053	100.00%	0.0010
30	120	86.75	0.0045	100.00%	0.0010

```
Elapsed time is 87.899947 seconds.
```

```
>> newnet = trainNetwork(net,data,options)


Training on single GPU.
Initializing image normalization.
|=======================================================================================================|
|     Epoch     |   Iteration   | Time Elapsed |  Mini-batch   |  Mini-batch   | Base Learning|
|               |               |  (seconds)   |     Loss      |   Accuracy    |     Rate     |
|=======================================================================================================|
|             1 |             1 |         0.47 |        3.5061 |         7.81% |       0.0010 |
|             3 |            10 |        10.31 |        0.7686 |        75.00% |       0.0010 |
|             5 |            20 |        18.96 |        0.2371 |        92.19% |       0.0010 |
|             8 |            30 |        27.43 |        0.0770 |        97.66% |       0.0010 |
|            10 |            40 |        35.31 |        0.0336 |        99.22% |       0.0010 |
|            13 |            50 |        43.17 |        0.0289 |        99.22% |       0.0010 |
|            15 |            60 |        50.15 |        0.0104 |       100.00% |       0.0010 |
|            18 |            70 |        56.84 |        0.0072 |       100.00% |       0.0010 |
|            20 |            80 |        63.00 |        0.0210 |        99.22% |       0.0010 |
|            23 |            90 |        69.37 |        0.0035 |       100.00% |       0.0010 |
|            25 |           100 |        74.85 |        0.0027 |       100.00% |       0.0010 |
|            28 |           110 |        81.19 |        0.0053 |       100.00% |       0.0010 |
|            30 |           120 |        86.75 |        0.0045 |       100.00% |       0.0010 |
|=======================================================================================================|
Elapsed time is 87.899947 seconds.
```

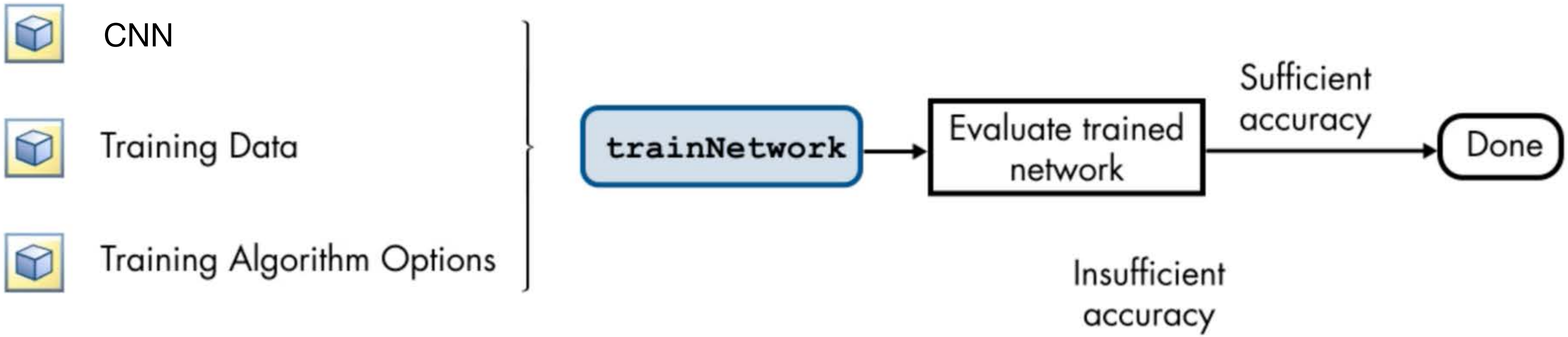
```
>> newnet = trainNetwork(net,data,options)
```

```
Training on single GPU.
Initializing image normalization.
```

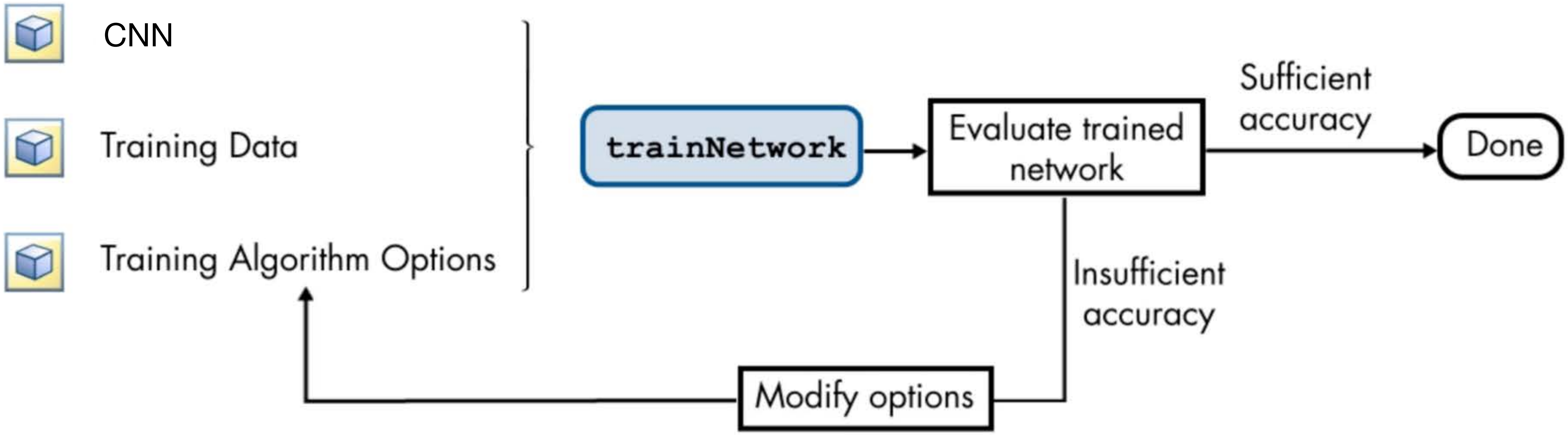
Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.47	3.5061	7.81%	0.0010
3	10	10.31	0.7686	75.00%	0.0010
5	20	18.96	0.2371	92.19%	0.0010
8	30	27.43	0.0770	97.66%	0.0010
10	40	35.31	0.0336	99.22%	0.0010
13	50	43.17	0.0289	99.22%	0.0010
15	60	50.15	0.0104	100.00%	0.0010
18	70	56.84	0.0072	100.00%	0.0010
20	80	63.00	0.0210	99.22%	0.0010
23	90	69.37	0.0035	100.00%	0.0010
25	100	74.85	0.0027	100.00%	0.0010
28	110	81.19	0.0053	100.00%	0.0010
30	120	86.75	0.0045	100.00%	0.0010

```
Elapsed time is 87.899947 seconds.
```

```
>> newnet = trainNetwork(net,data,options)


Training on single GPU.
Initializing image normalization.
|========================================================================================|
|     Epoch     |   Iteration   | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|               |               |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|========================================================================================|
|             1 |             1 |         0.47 |       3.5061 |        7.81% |       0.0010 |
|             3 |            10 |        10.31 |       0.7686 |       75.00% |       0.0010 |
|             5 |            20 |        18.96 |       0.2371 |       92.19% |       0.0010 |
|             8 |            30 |        27.43 |       0.0770 |       97.66% |       0.0010 |
|            10 |            40 |        35.31 |       0.0336 |       99.22% |       0.0010 |
|            13 |            50 |        43.17 |       0.0289 |       99.22% |       0.0010 |
|            15 |            60 |        50.15 |       0.0104 |      100.00% |       0.0010 |
|            18 |            70 |        56.84 |       0.0072 |      100.00% |       0.0010 |
|            20 |            80 |        63.00 |       0.0210 |       99.22% |       0.0010 |
|            23 |            90 |        69.37 |       0.0035 |      100.00% |       0.0010 |
|            25 |           100 |        74.85 |       0.0027 |      100.00% |       0.0010 |
|            28 |           110 |        81.19 |       0.0053 |      100.00% |       0.0010 |
|            30 |           120 |        86.75 |       0.0045 |      100.00% |       0.0010 |
|========================================================================================|
Elapsed time is 87.899947 seconds.
```



CNN
Training Data
Training Algorithm Options
trainNetwork
Evaluate trained network
Sufficient accuracy
Done
Insufficient accuracy


CNN
Training Data
Training Algorithm Options
trainNetwork
Evaluate trained network
Sufficient accuracy
Done
Insufficient accuracy
Modify options

Training Algorithm Options

`InitialLearnRate`

`Momentum`

4. Test the performance of the CNN

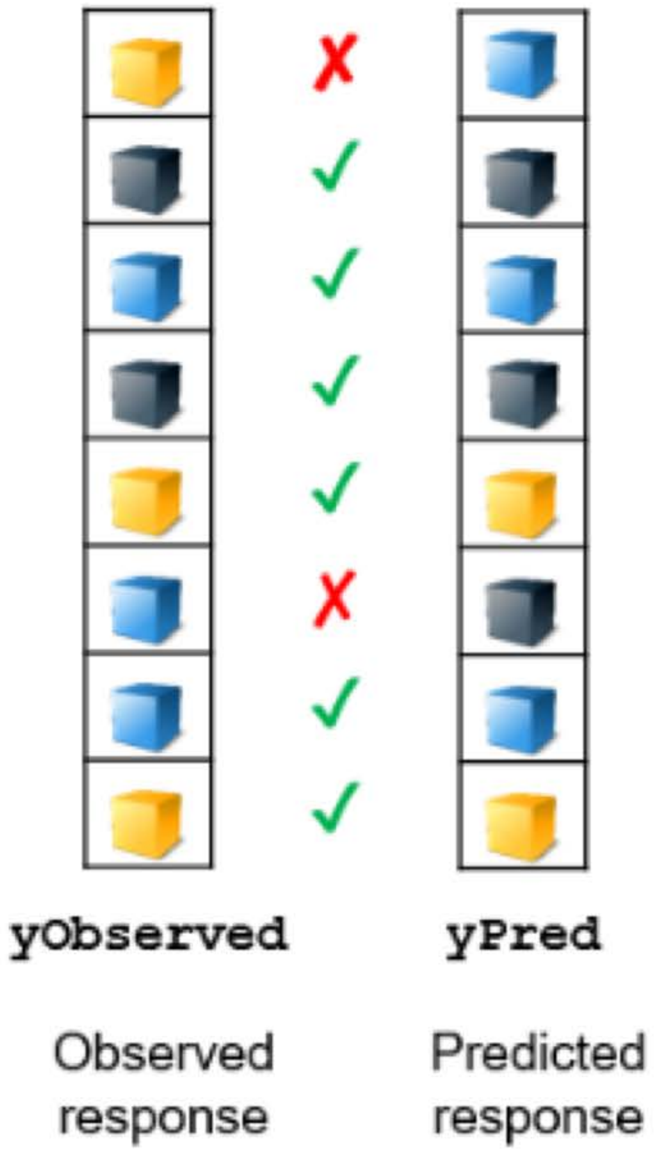
```
%% Test the performance of the NN

% test network performance on validation set
[labels,~] = classify(XONet, validationDS, 'MiniBatchSize', 128);

% calculate the confusion matrix.
confMat = confusionmat(validationDS.Labels, labels);
confMat = bsxfun(@rdivide,confMat,sum(confMat,2));
fprintf('Performance on validation set \t\t\t%.4f\n',mean(diag(confMat)));
```

Obtain predictions on the validationDS

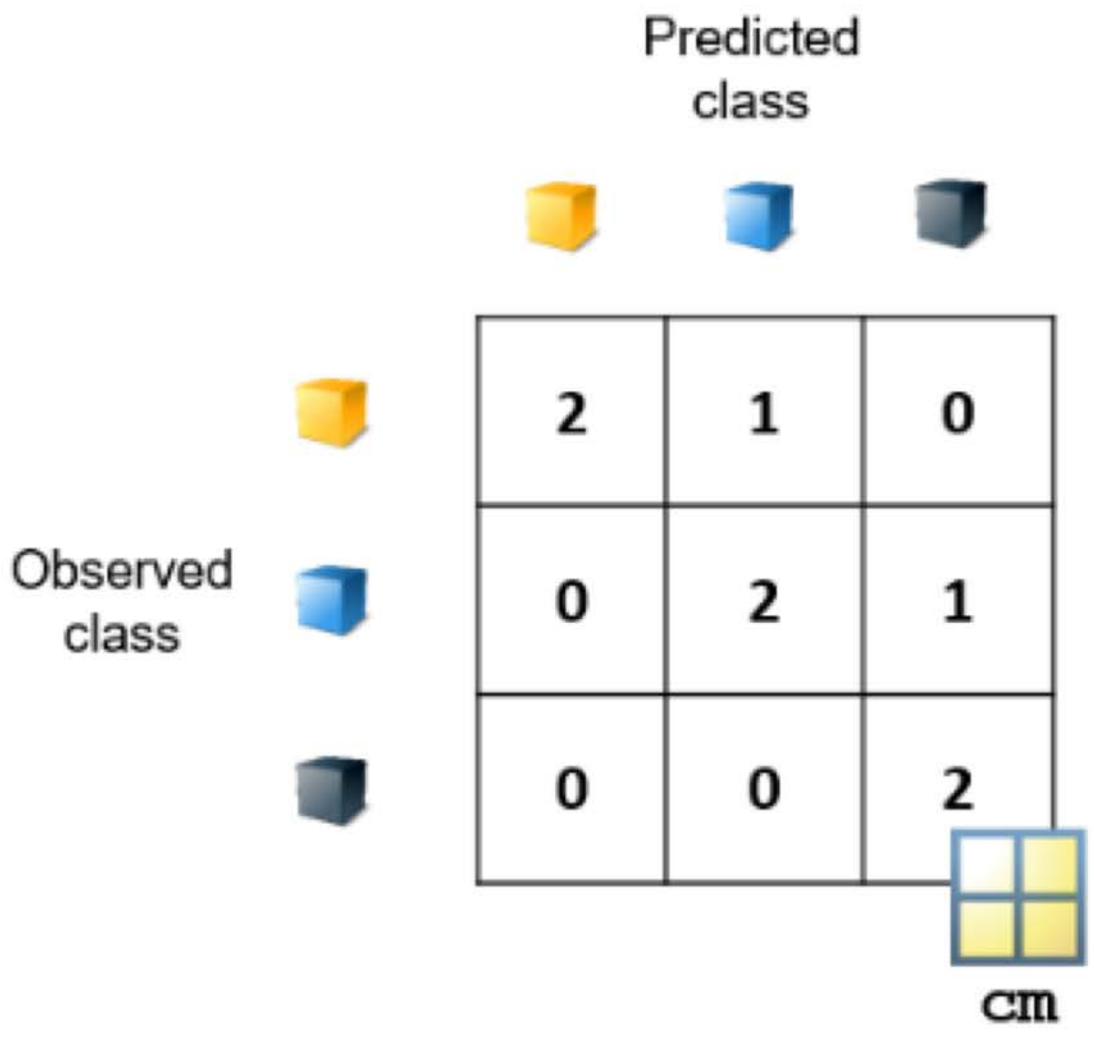
Compute the confusion matrix

Report the mean accuracy

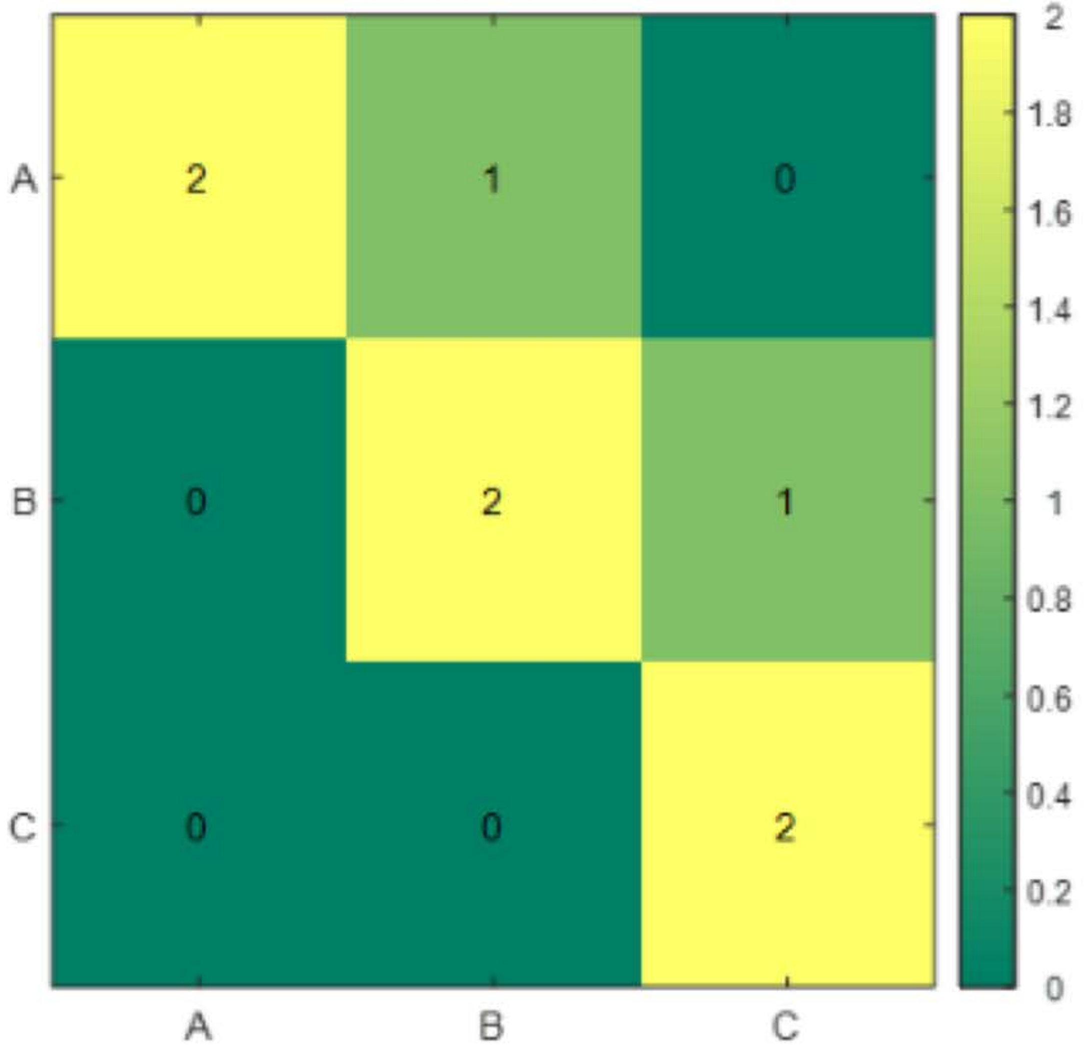
```
>> [cm,grp] = confusionmat(yObserved,yPred)
```



```
>> [cm,grp] = confusionmat(yObserved,yPred)
cm =
    2  1  0
    0  2  1
    0  0  2
grp =
    A
    B
    C
```



```
>> [cm,grp] = confusionmat(yObserved,yPred)
cm =
    2   1   0
    0   2   1
    0   0   2
grp =
    A
    B
    C
>> heatmap(cm,grp,grp,true,...
    'Colormap','summer',...
    'Colorbar',true)
```



5. Plotting options

1. Plot wrongly classifies images from the ValidationDS
2. Plot the filters from the Convolution layers
3. Plot the feature maps for some of the input images

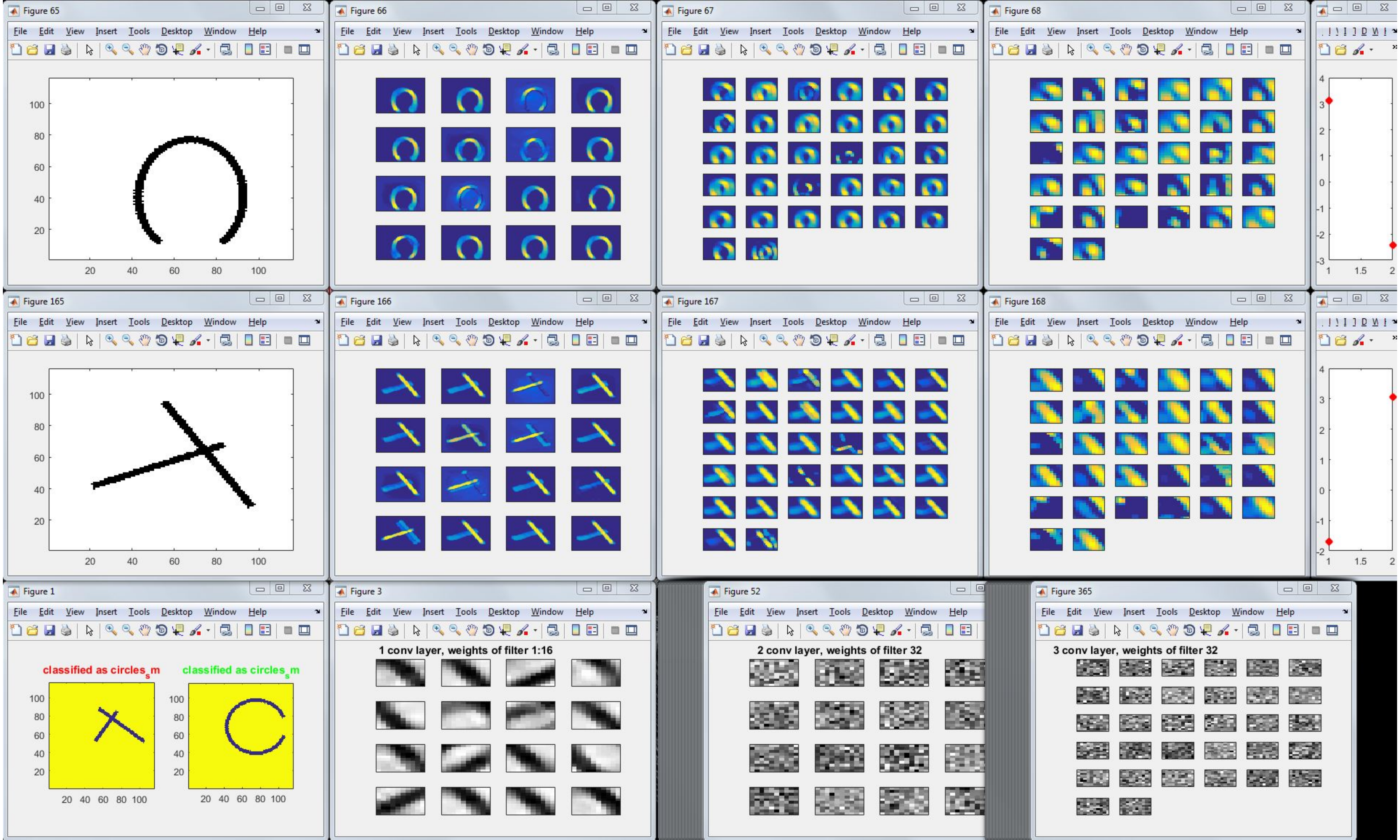
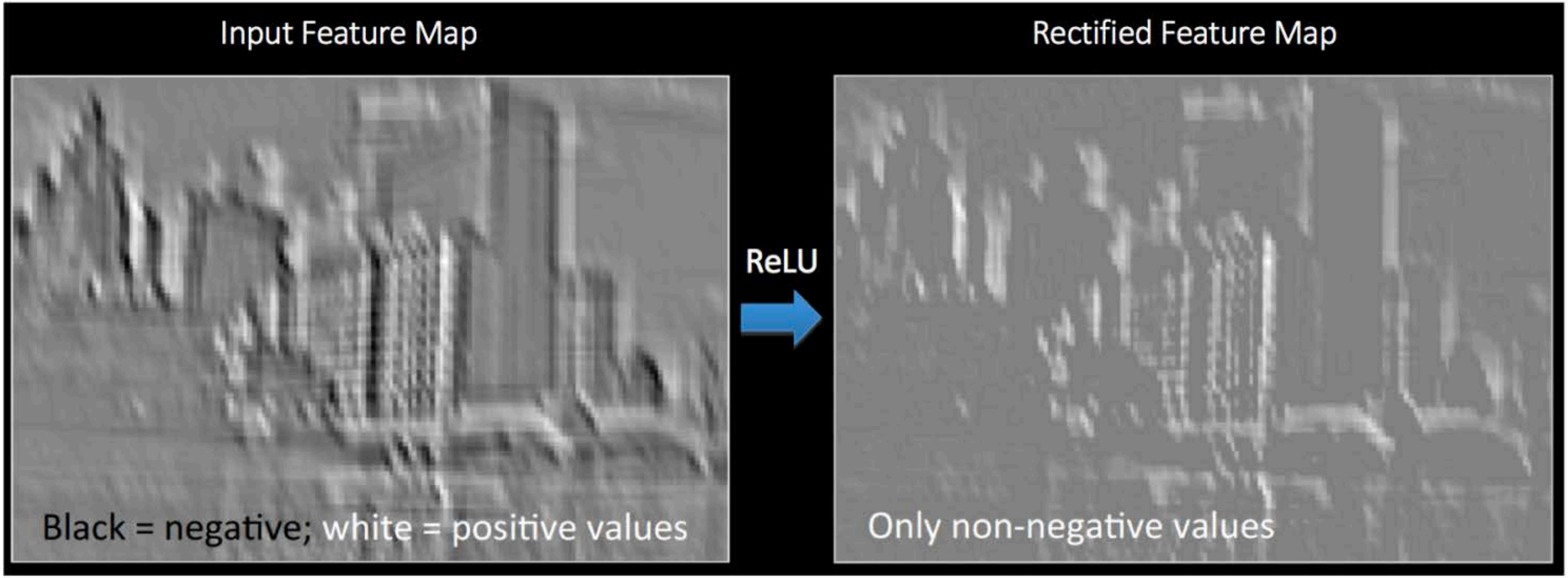
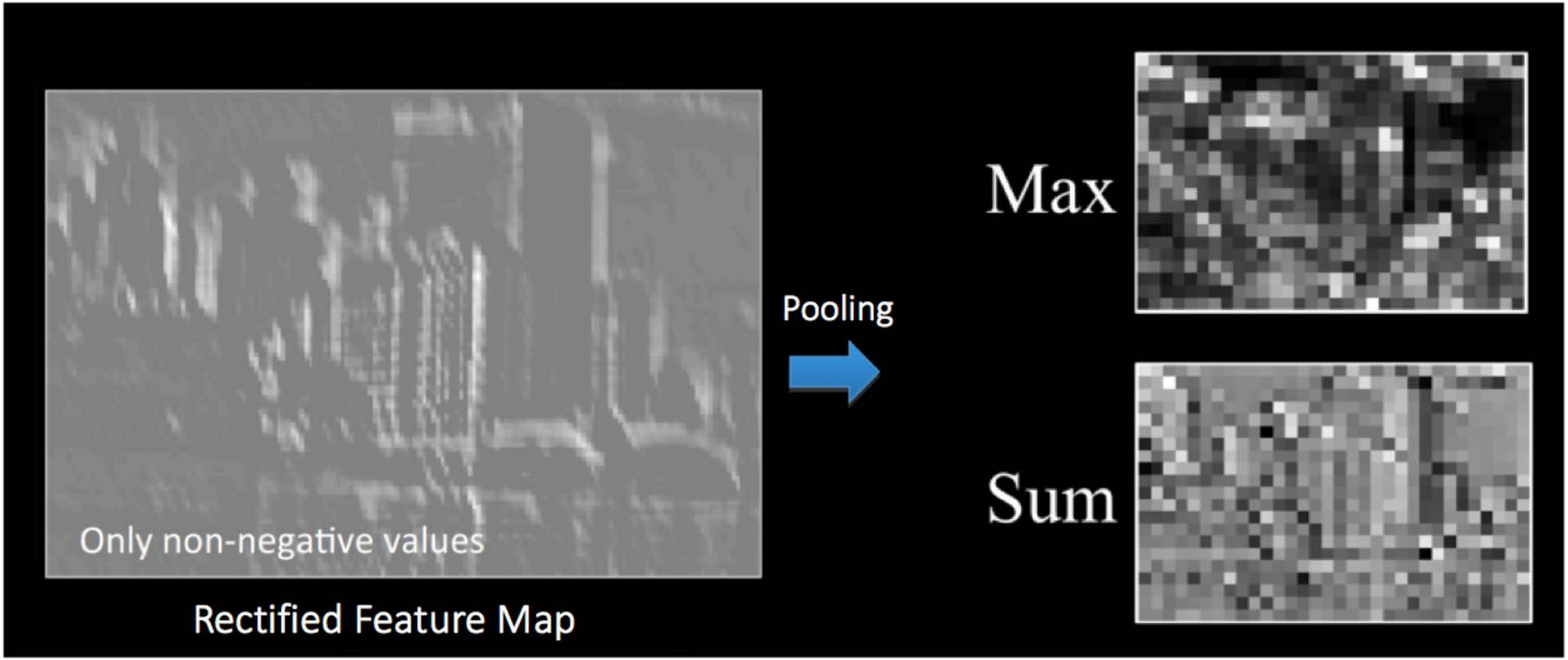
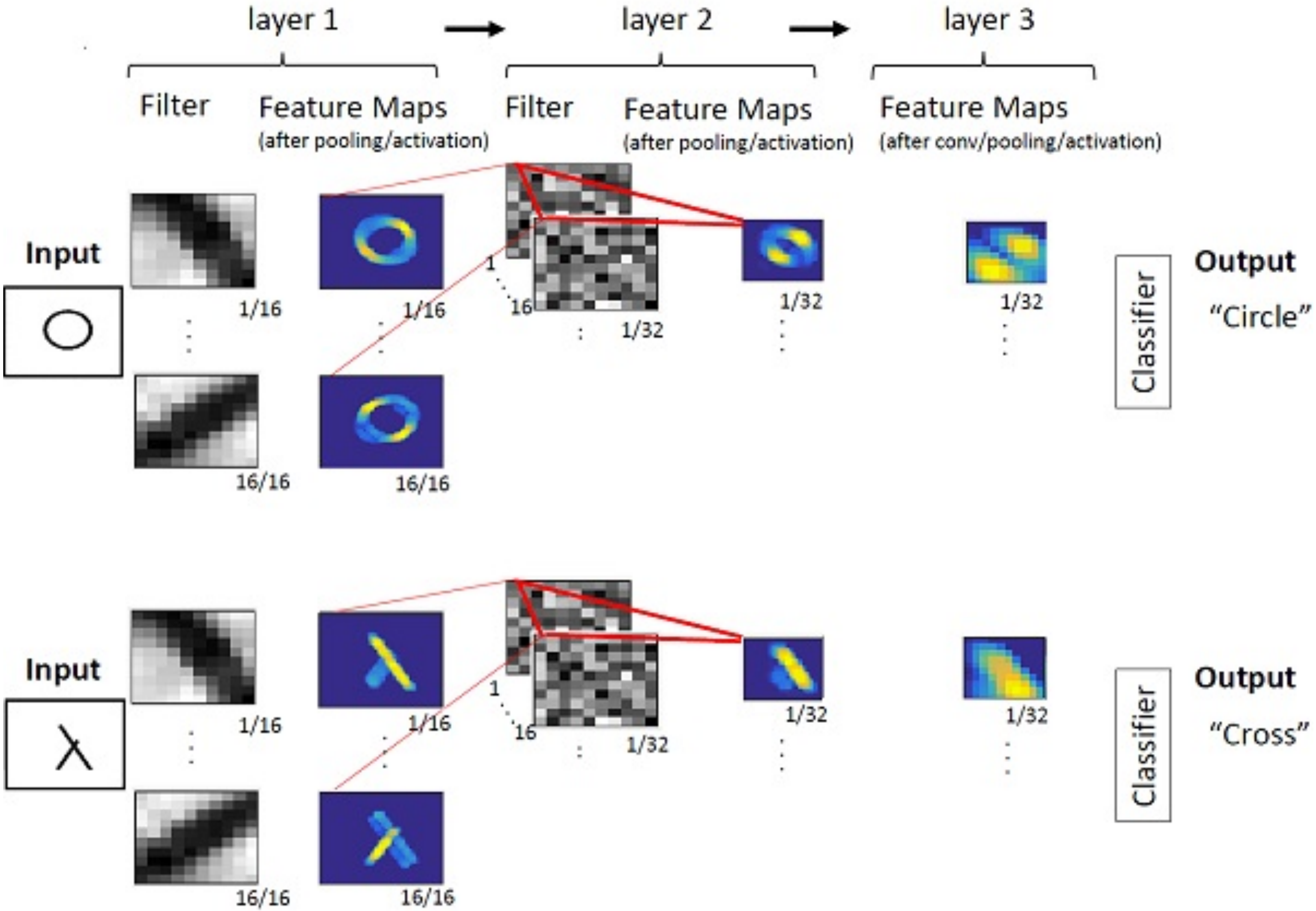


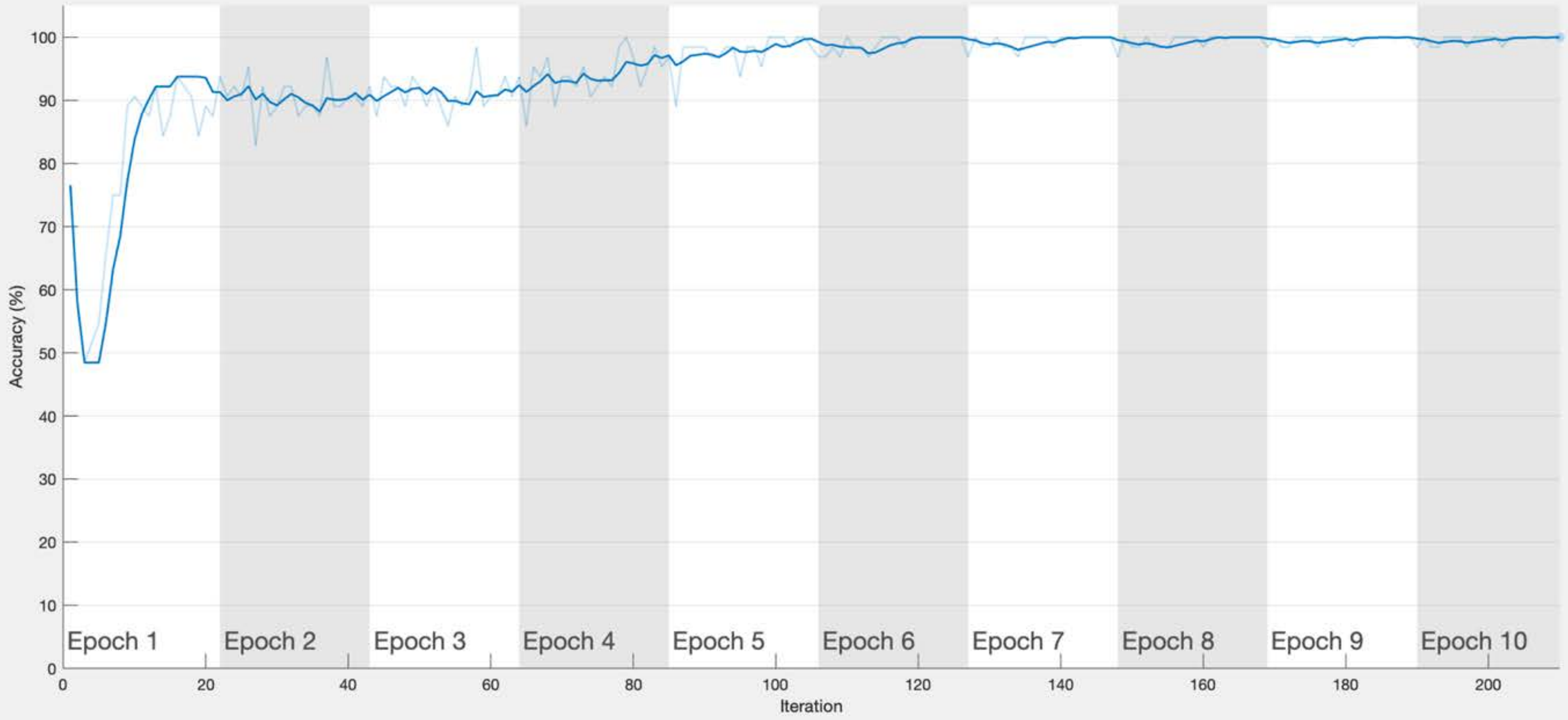
Figure 65
Figure 66
Figure 67
Figure 68
Figure 165
Figure 166
Figure 167
Figure 168
Figure 1
classified as circles_m
classified as circles_m
Figure 3
1 conv layer, weights of filter 1:16
Figure 52
2 conv layer, weights of filter 32
Figure 365
3 conv layer, weights of filter 32

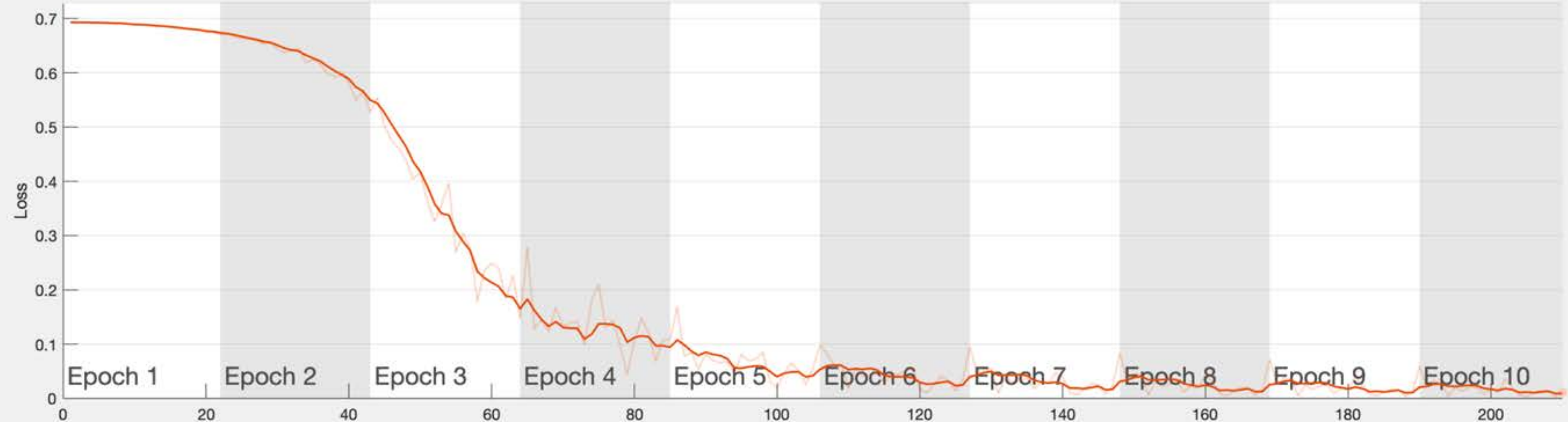

Input


Input Feature Map
Rectified Feature Map
ReLU
Black = negative; white = positive values
Only non-negative values


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum


layer 1
layer 2
layer 3
Filter
Feature Maps
(after pooling/activation)
Filter
Feature Maps
(after pooling/activation)
Feature Maps
(after conv/pooling/activation)
Input
1/16
16/16
1/16
16/16
1
16
1/32
1/32
1/32
Classifier
Output
"Circle"
Input
1/16
16/16
1/16
16/16
1
16
1/32
1/32
1/32
Classifier
Output
"Cross"





Results

Validation accuracy: N/A
Training finished: Reached final iteration

Training Time

Start time: 20-Nov-2019 14:14:55
Elapsed time: 6 min 25 sec

Training Cycle

Epoch: 10 of 10
Iteration: 210 of 210
Iterations per epoch: 21
Maximum iterations: 210

Validation

Frequency: N/A
Patience: N/A

Other Information

Hardware resource: Single CPU
Learning rate schedule: Piecewise
Learning rate: 0.001

Learn more

Accuracy

Training (smoothed)
Training
Validation

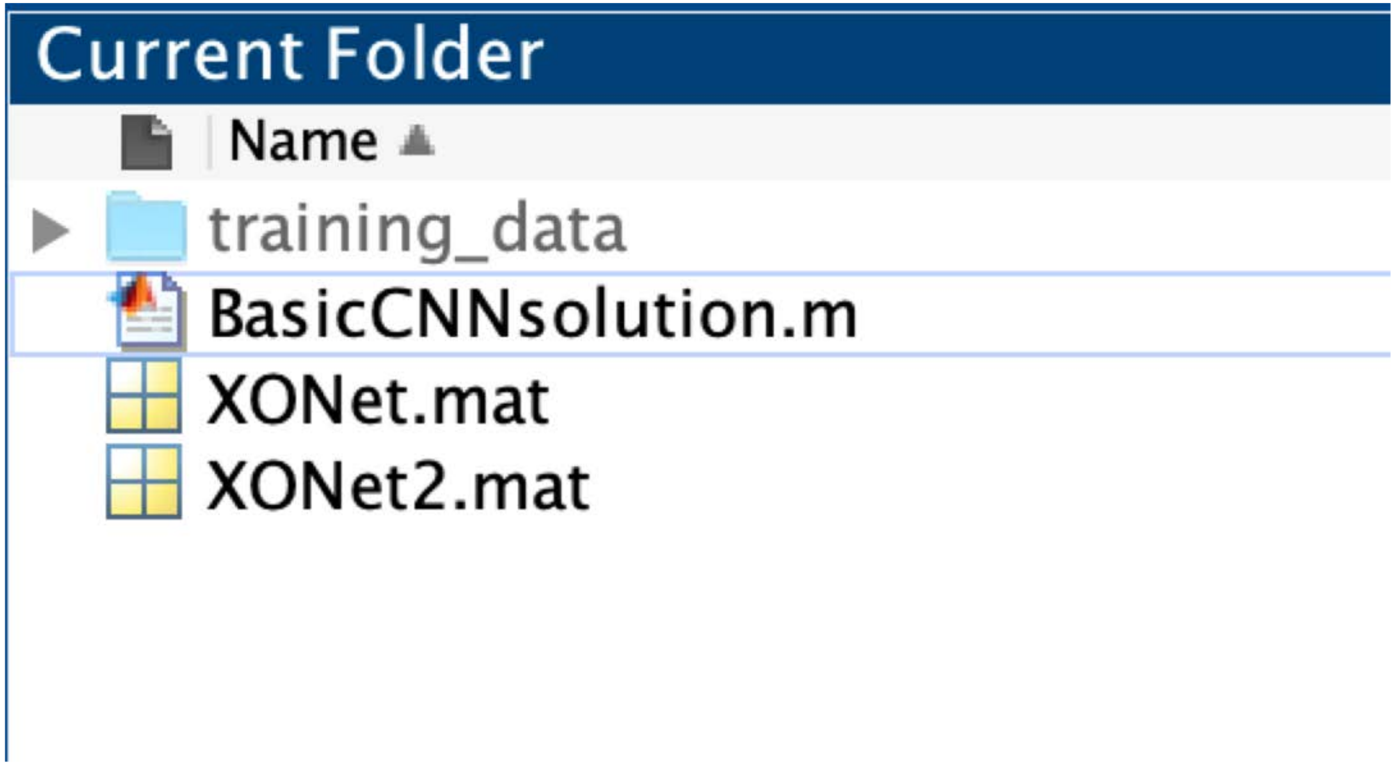
Loss

Training (smoothed)
Training

You need to submit:

1. **Upload a single .zip file** with the filename [firstname_lastname_UVA_computing_ID].zip
2. The zip file should contain:
    a. The original training_data folder with the subfolders circles and crosses with the images included (this is < 3.6 Mb)
    b. Your solution to BasicCNNtemplate.m
    c. Your best performing network in the form of a .mat file XONet (This file is automatically created when doTraining == true)
    d. Your responses to the effect of Momentum, InitialTraiingRate, and Epochs on the performance of the network – Include supporting plots and accuracy values.
        i. This can be a PDF with the plots and inferences included.
    e. Report (with plots) on the architecture, and accuracy of your best performing network:
        i. Include an image of the layers of the network.
        ii. Report accuracy (as computed by the template, using the confusion matrix) on the validationDS of your best performing model.
        iii. Report the chosen values of the hyperparameters of your network.

Submit your best model as a .mat file



Not mandatory to use Matlab: [Part 1]

1. Use whatever DL framework you are familiar/comfortable with.

2. Provide all your code and include a 'requirements.txt' file to list all the dependencies needed to run the code. [https://pip.readthedocs.io/en/1.1/requirements.html]

3. You are responsible for generating all the plots required by the assignment.

Not mandatory to use Matlab: [Part 2]

1. Must provide the best performing CNN as a .mat file

2. Use Open Neural Network Exchange (ONNX) standard.

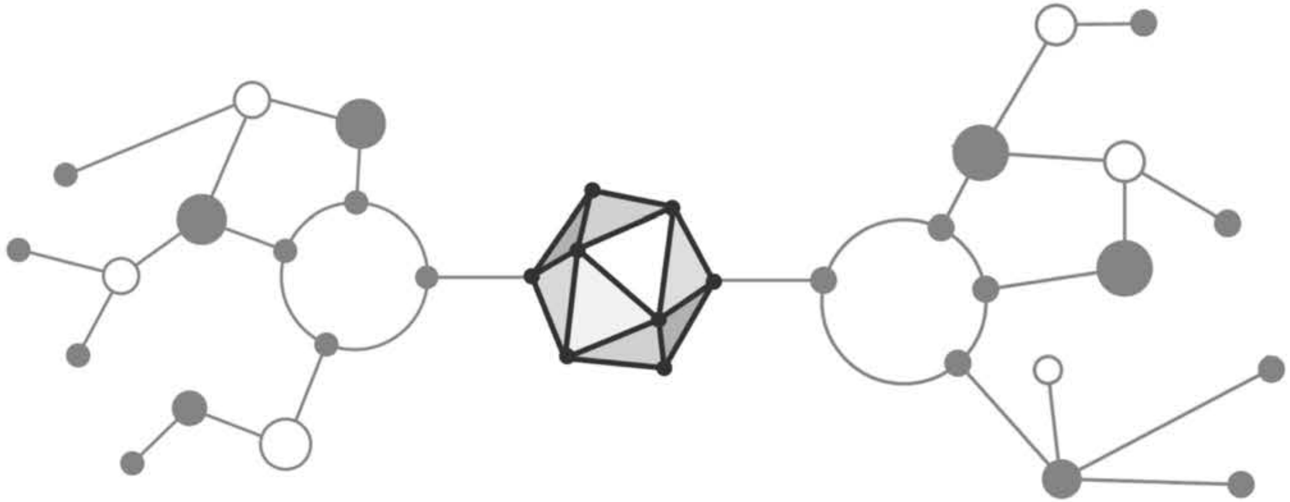
Open Neural Network Exchange (ONNX)

1. Export your CNN from your framework as a ONNX model. Examples:

   https://github.com/onnx/tutorials

2. Use importONNXNetwork in Matlab and generate the .mat file

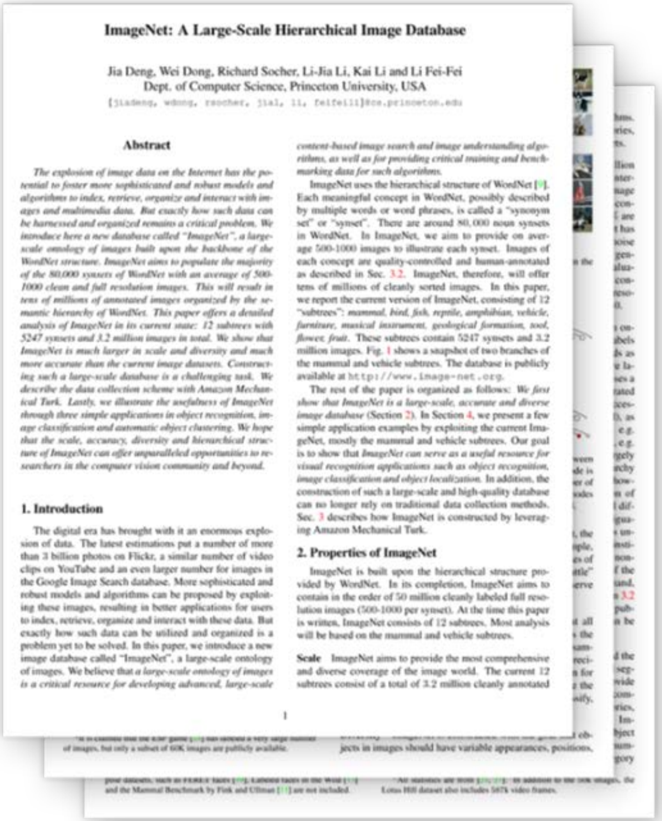
ONNX



IMAGENET

How a dataset changed deep learning

The Beginning: CVPR 2009


ImageNet: A Large-Scale Hierarchical Image Database

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei
Dept. of Computer Science, Princeton University, USA

Abstract

1. Introduction

2. Properties of ImageNet

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ***ImageNet: A Large-Scale Hierarchical Image Database.***
IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

IMAGENET on Google Scholar

4,386
Citations

[Imagenet: A large-scale hierarchical image database](#)
J Deng, W Dong, R Socher, LJ Li, K Li… - Computer Vision and …, 2009 - ieeexplore.ieee.org
Abstract: The explosion of image data on the Internet has the potential to foster more sophisticated and robust models and algorithms to index, retrieve, organize and interact with images and multimedia data. But exactly how such data can be harnessed and organized
Cited by 4386 Related articles All 30 versions Cite Save

2,847
Citations

Imagenet large scale visual recognition challenge
O Russakovsky, J Deng, H Su, J Krause… - International Journal of …, 2015 - Springer
Abstract The **ImageNet** Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on hundreds of object categories and millions of images. The challenge has been run annually from 2010 to present, attracting participation
Cited by 2847 Related articles All 17 versions Cite Save

…and many more.

From IMAGENET Challenge Contestants to Startups

VizSense®

ViSENZE
Simplifying the Visual Web

clarifai

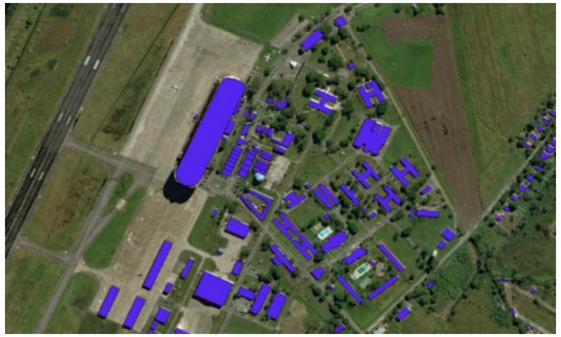
Lunit
Toward Data-Driven Medicine

MetaMind

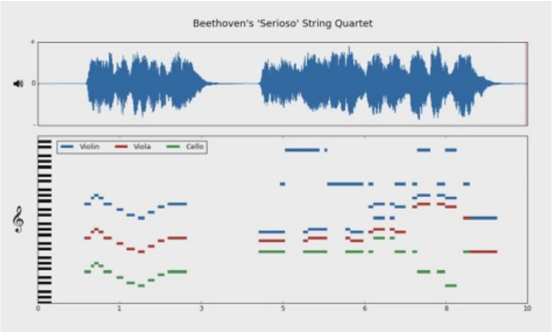
DNNresearch

VUNO

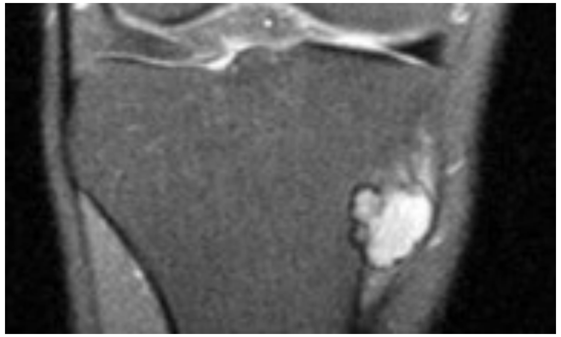
“The IMAGENET of x”



SpaceNet
DigitalGlobe, CosmiQ Works, NVIDIA



MusicNet
J. Thickstun et al, 2017



Medical ImageNet
Stanford Radiology, 2017

ShapeNet
A.Chang et al, 2015

EventNet
G. Ye et al, 2015

ActivityNet
F. Heilbron et al, 2015

Hardly the First Image Dataset

Segmentation (2001)
D. Martin, C. Fowlkes, D. Tal, J. Malik.

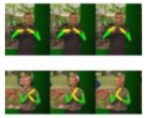
CMU/VASC Faces (1998)
H. Rowley, S. Baluja, T. Kanade

FERET Faces (1998)
P. Phillips, H. Wechsler, J. Huang, P. Raus

COIL Objects (1996)
S. Nene, S. Nayar, H. Murase

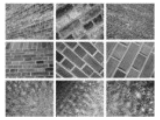
MNIST digits (1998-10)
Y LeCun & C. Cortes

KTH human action (2004)
I. Leptev & B. Caputo



Sign Language (2008)
P. Buehler, M. Everingham, A. Zisserman

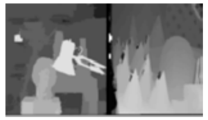
UIUC Cars (2004)
S. Agarwal, A. Awan, D. Roth



3D Textures (2005)
S. Lazebnik, C. Schmid, J. Ponce

CuRRET Textures (1999)
K. Dana B. Van Ginneken S. Nayar J. Koenderink

CAVIAR Tracking (2005)
R. Fisher, J. Santos-Victor J. Crowley



Middlebury Stereo (2002)
D. Scharstein R. Szeliski

CalTech 101/256 (2005)
Fei-Fei et al, 2004
GriffIn et al, 2007

LabelMe (2005)
Russell et al, 2005

ESP (2006)
Ahn et al, 2006

MSRC (2006)
Shotton et al. 2006

PASCAL (2007)
Everingham et al, 2009

Lotus Hill (2007)
Yao et al, 2007

TinyImage (2008)
Torralba et al. 2008

A new way of thinking…

To shift the focus of Machine Learning for visual recognition

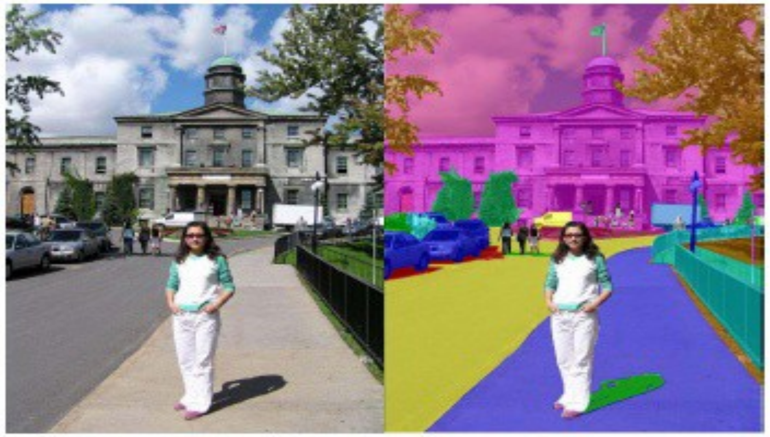
from modeling…

…to data.
Lots of data.

While Others Targeted Detail...

LabelMe

Per-Object Regions and Labels
Russell et al, 2005



Lotus Hill

Hand-Traced Parse Trees
Yao et al, 2007

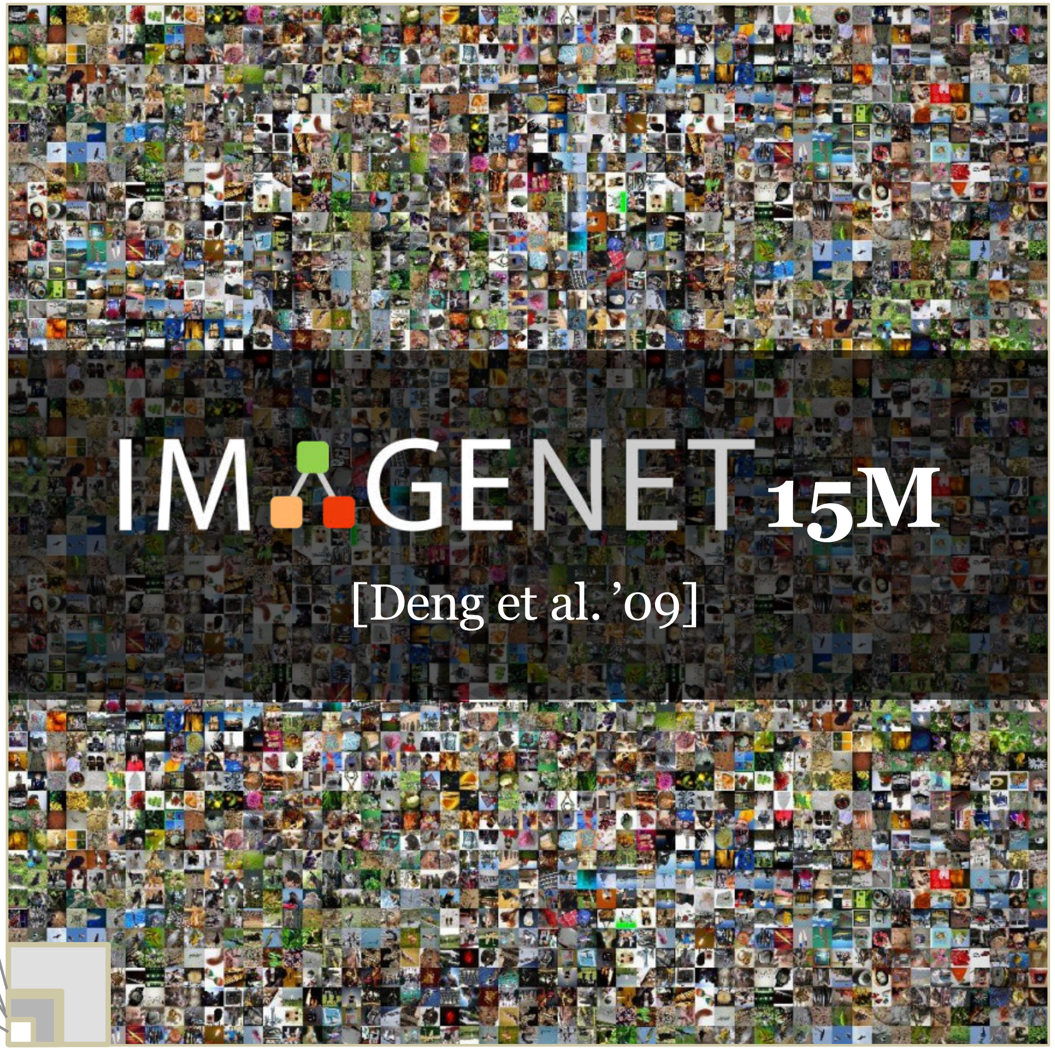
…ImageNet Targeted Scale

SUN, 131K
[Xiao et al. ‘10]

LabelMe, 37K
[Russell et al. ’07]

PASCAL VOC, 30K
[Everingham et al. ’06-’12]

Caltech101, 9K
[Fei-Fei, Fergus, Perona, ‘03]



IMAGENET Goals

High Resolution

To better replicate human visual acuity

Carnivore
- Canine
  - Dog
    - Working Dog
      - Husky

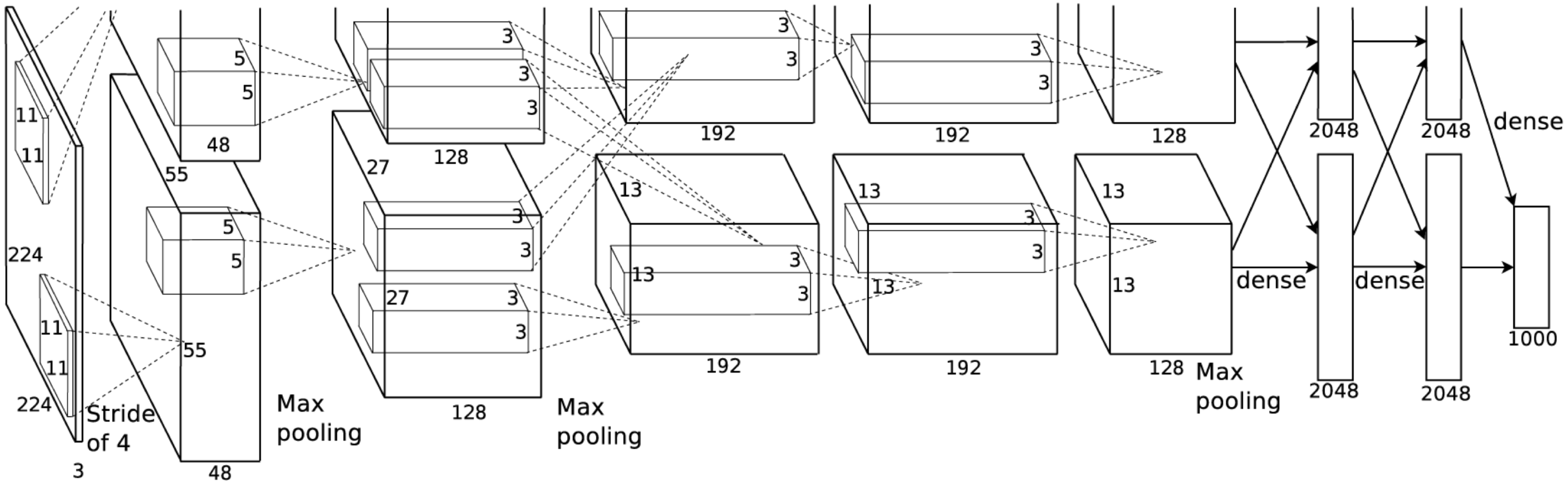
High-Quality Annotation

To create a benchmarking dataset and advance the state of machine perception, not merely reflect it

Free of Charge

To ensure immediate application and a sense of community

Neural Nets are Cool Again!



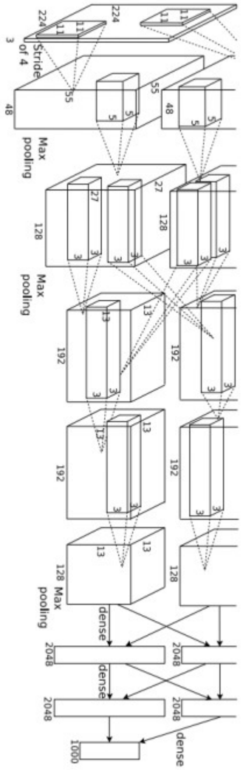
13,259
Citations

Imagenet classification with deep convolutional neural networks
A Krizhevsky, I Sutskever, GE Hinton - Advances in neural …, 2012 - papers.nips.cc
Abstract We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 **ImageNet** training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\%
Cited by 13259 Related articles All 95 versions Cite Save

Krizhevsky, Sutskever & Hinton, NIPS 2012

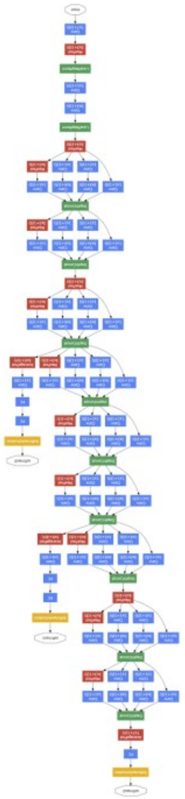
…And Cooler and Cooler J

"AlexNet"



[Krizhevsky et al. NIPS 2012]

"GoogLeNet"

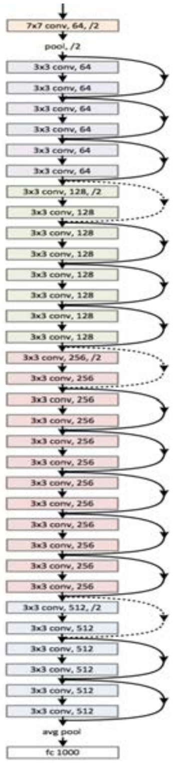


[Szegedy et al. CVPR 2015]

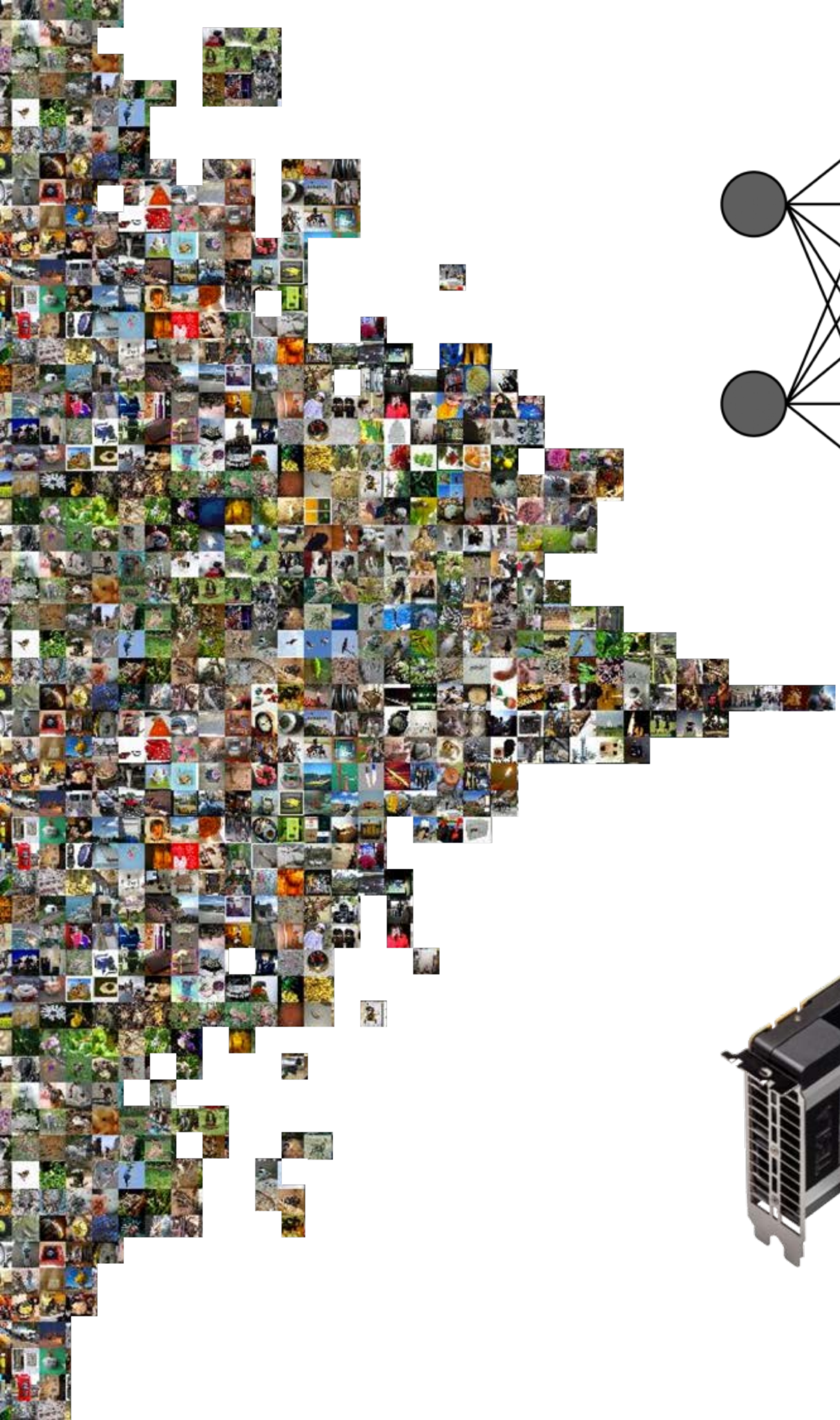
"VGG Net"

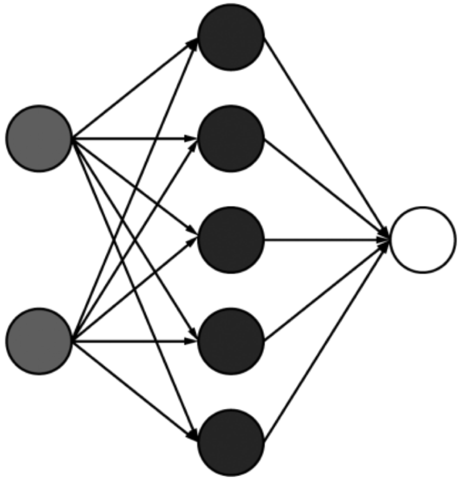
[Simonyan & Zisserman, ICLR 2015]

"ResNet"



[He et al. CVPR 2016]





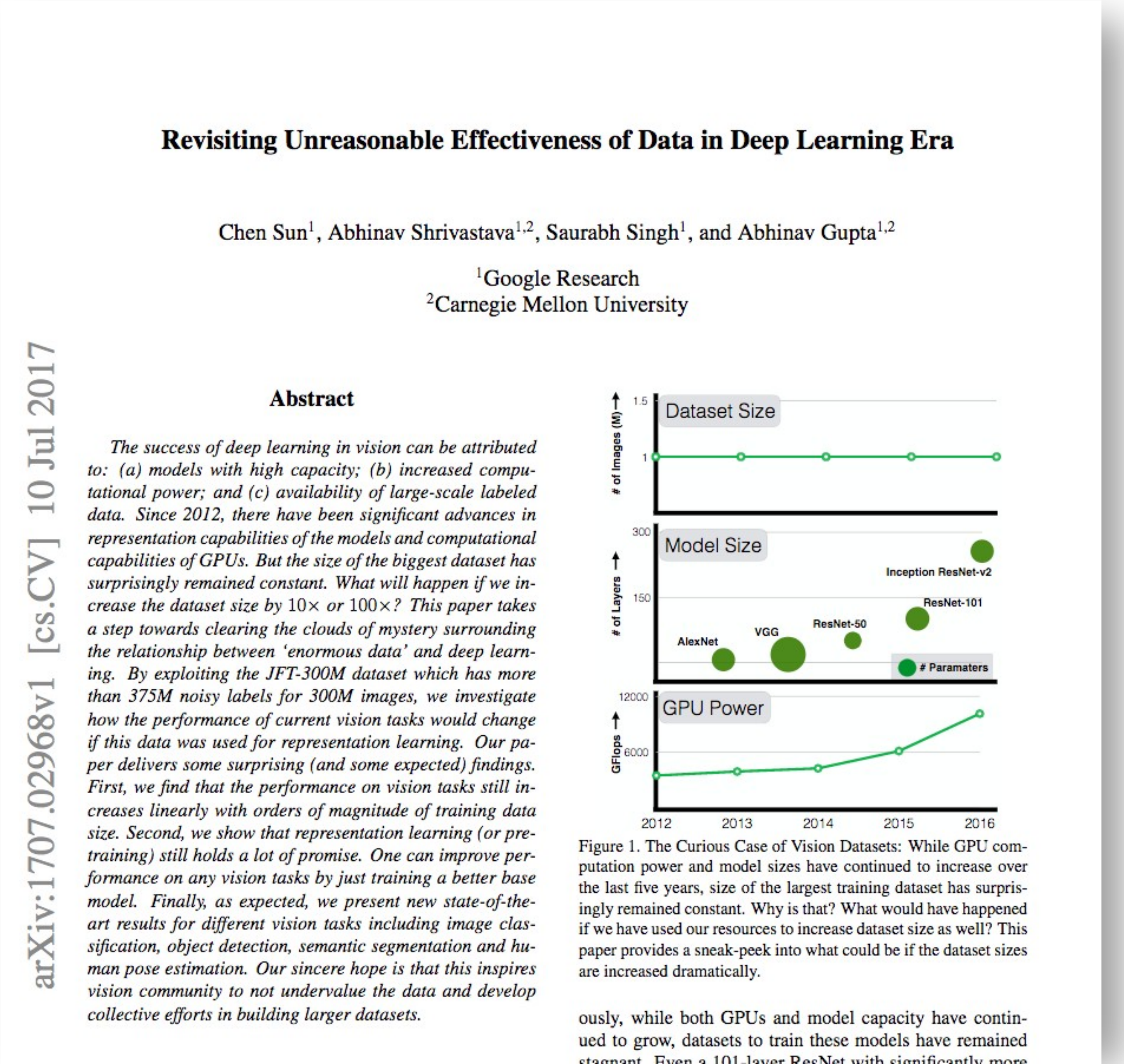
Neural Nets

GPUs

A Deep Learning Revolution

“First, we find that the performance on vision tasks still increases linearly with orders of magnitude of training data size.”

C. Sun et al, 2017



Revisiting Unreasonable Effectiveness of Data in Deep Learning Era

Chen Sun[1], Abhinav Shrivastava[1,2], Saurabh Singh[1], and Abhinav Gupta[1,2]

[1]Google Research
[2]Carnegie Mellon University

arXiv:1707.02968v1 [cs.CV] 10 Jul 2017

Abstract

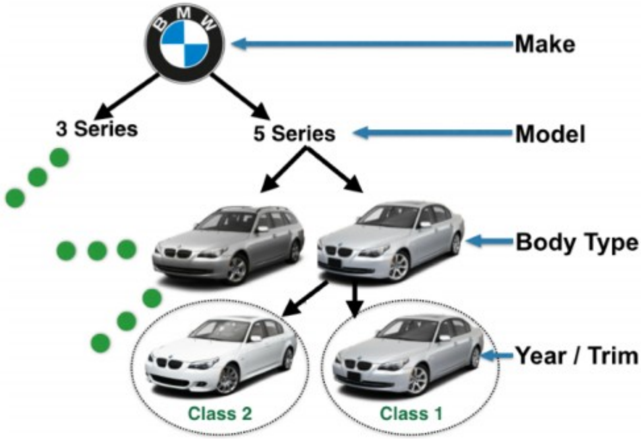
The success of deep learning in vision can be attributed to: (a) models with high capacity; (b) increased computational power; and (c) availability of large-scale labeled data. Since 2012, there have been significant advances in representation capabilities of the models and computational capabilities of GPUs. But the size of the biggest dataset has surprisingly remained constant. What will happen if we increase the dataset size by 10× or 100×? This paper takes a step towards clearing the clouds of mystery surrounding the relationship between ‘enormous data’ and deep learning. By exploiting the JFT-300M dataset which has more than 375M noisy labels for 300M images, we investigate how the performance of current vision tasks would change if this data was used for representation learning. Our paper delivers some surprising (and some expected) findings. First, we find that the performance on vision tasks still increases linearly with orders of magnitude of training data size. Second, we show that representation learning (or pretraining) still holds a lot of promise. One can improve performance on any vision tasks by just training a better base model. Finally, as expected, we present new state-of-the-art results for different vision tasks including image classification, object detection, semantic segmentation and human pose estimation. Our sincere hope is that this inspires vision community to not undervalue the data and develop collective efforts in building larger datasets.

Figure 1. The Curious Case of Vision Datasets: While GPU computation power and model sizes have continued to increase over the last five years, size of the largest training dataset has surprisingly remained constant. Why is that? What would have happened if we have used our resources to increase dataset size as well? This paper provides a sneak-peek into what could be if the dataset sizes are increased dramatically.

ously, while both GPUs and model capacity have continued to grow, datasets to train these models have remained

Fine-Grained Recognition

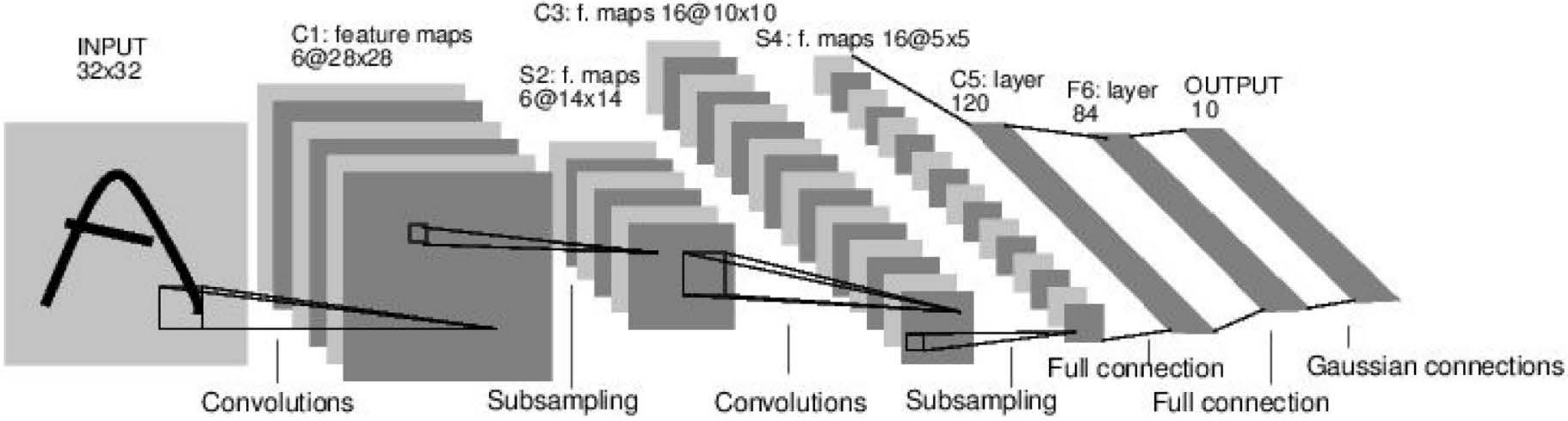
IMAGENET cars [Gebru, Krause, Deng, Fei-Fei, CHI 2017]



2567 classes
700k images

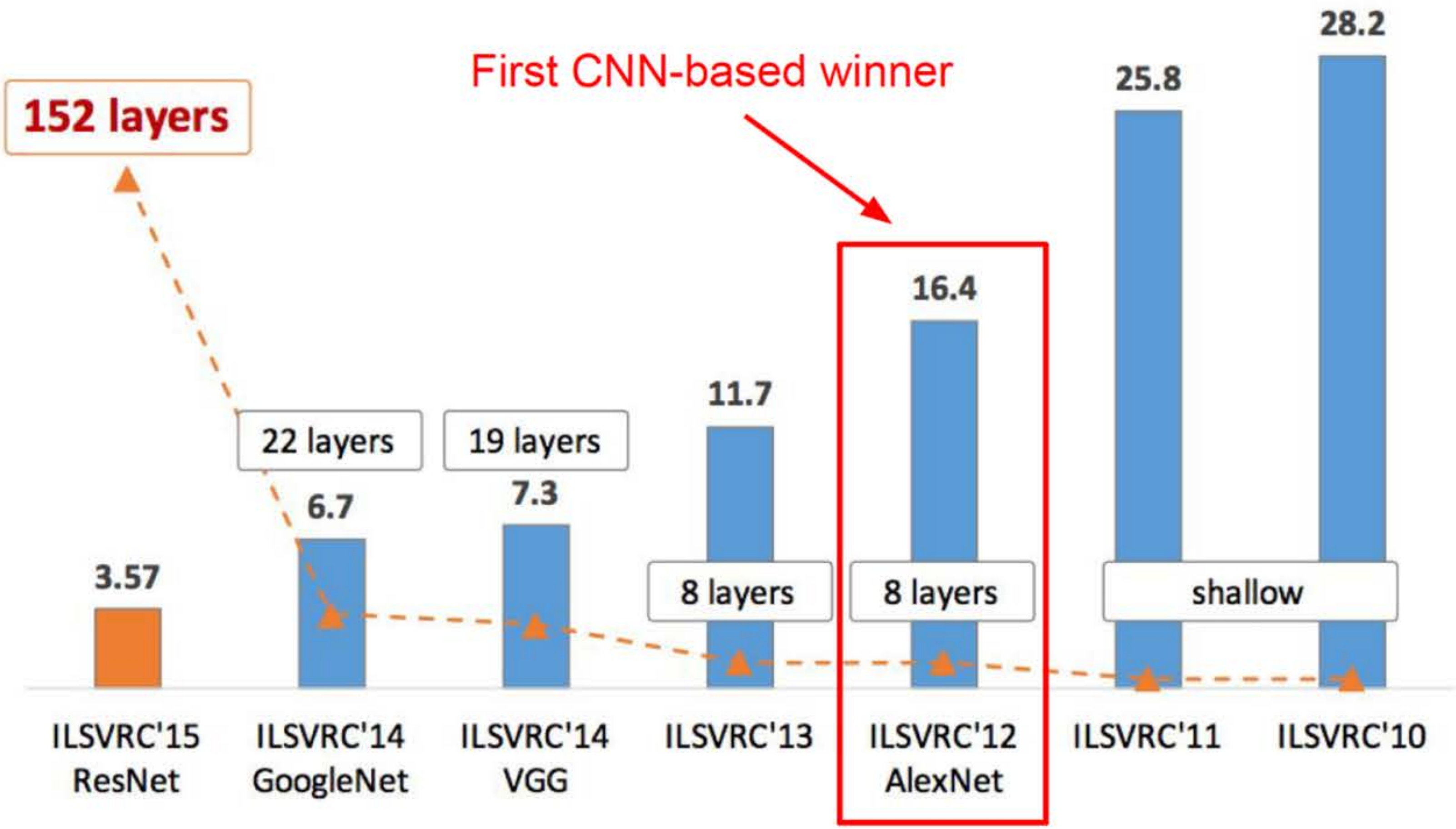
Case Study: LeNet-5

[LeCun et al., 1998]



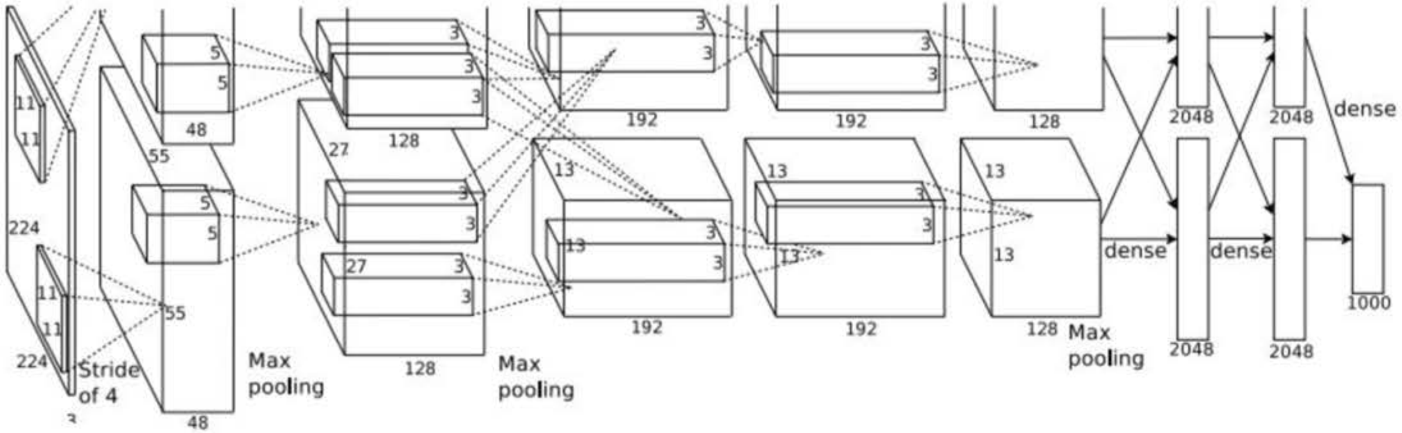
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: AlexNet

[Krizhevsky et al. 2012]

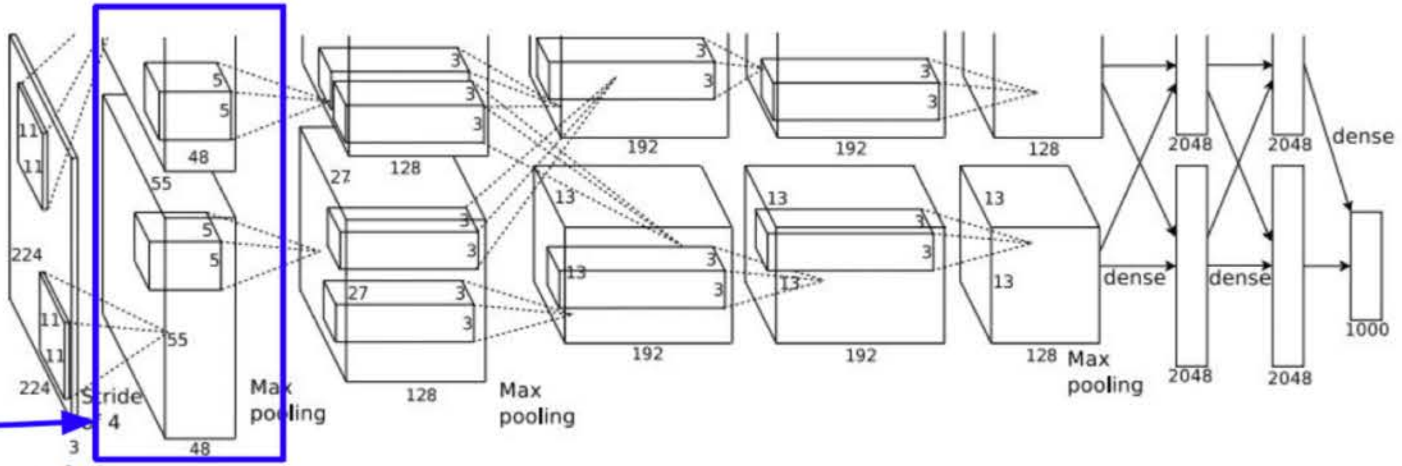


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
CONV: 96 11x11 filters at stride 4, pad 0
MAX POOL1: 3x3 filters at stride 2
NORM1: Normalization layer
CONV2: 265 5x5 filters at stride 1, pad 2
MAX POOL 2: 3x3 filters at stride 2
NORM2: Normalization layer
CONV3: 384 3x3 filters at stride 1, pad 1
CONV4: 384 3x3 filters at stride 1, pad 1
CONV5: 265 3x3 filters at stride 1, pad 1
MAX POOL 3: 3x3 filters at stride 2
FC6: Fully connected layer (4096 neurons)
FC7: Fully connected layer (4096 neurons)
FC8: 1000 neurons (logit scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: AlexNet

[Krizhevsky et al. 2012]

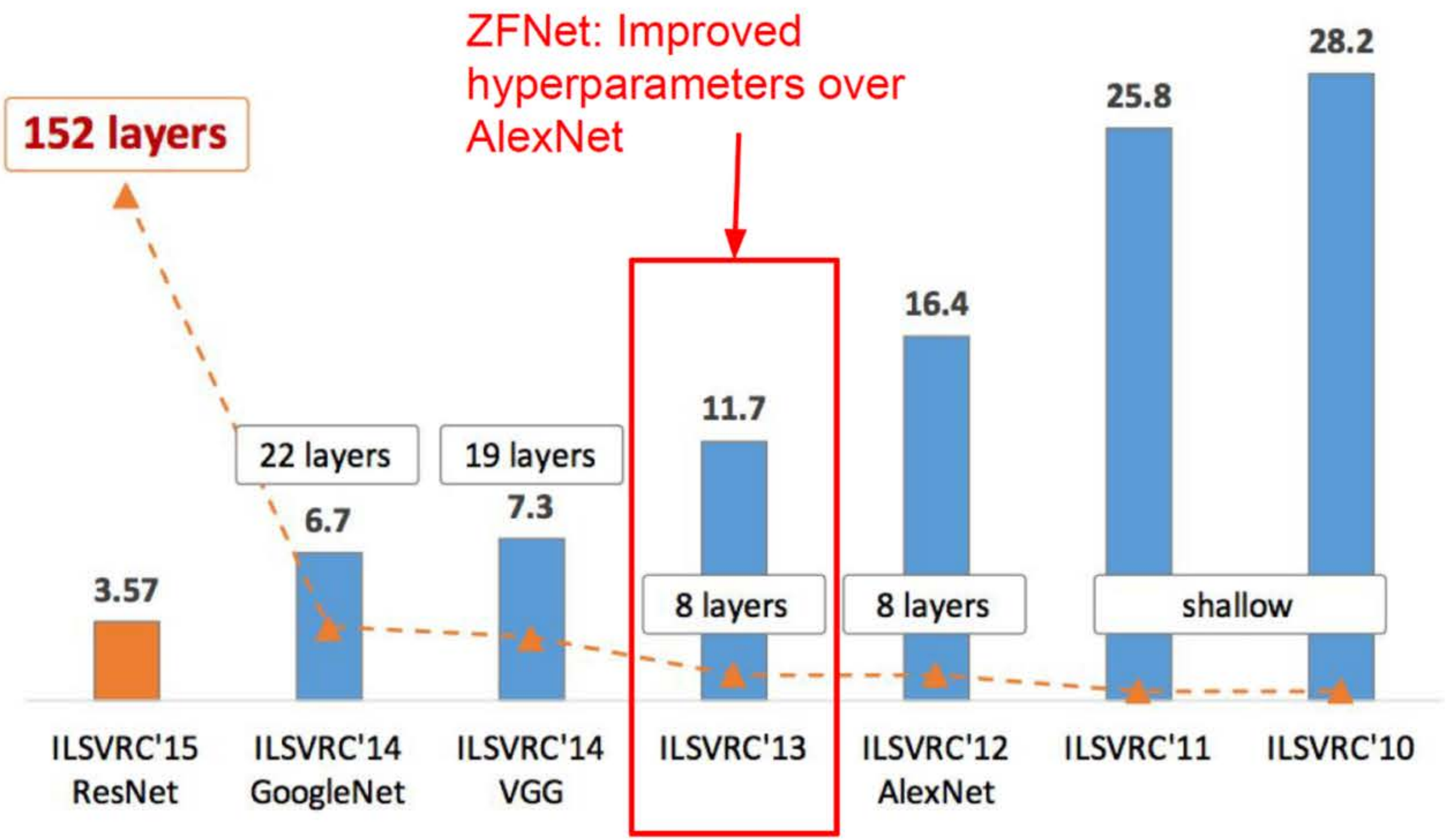


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

[55x55x48] x 2

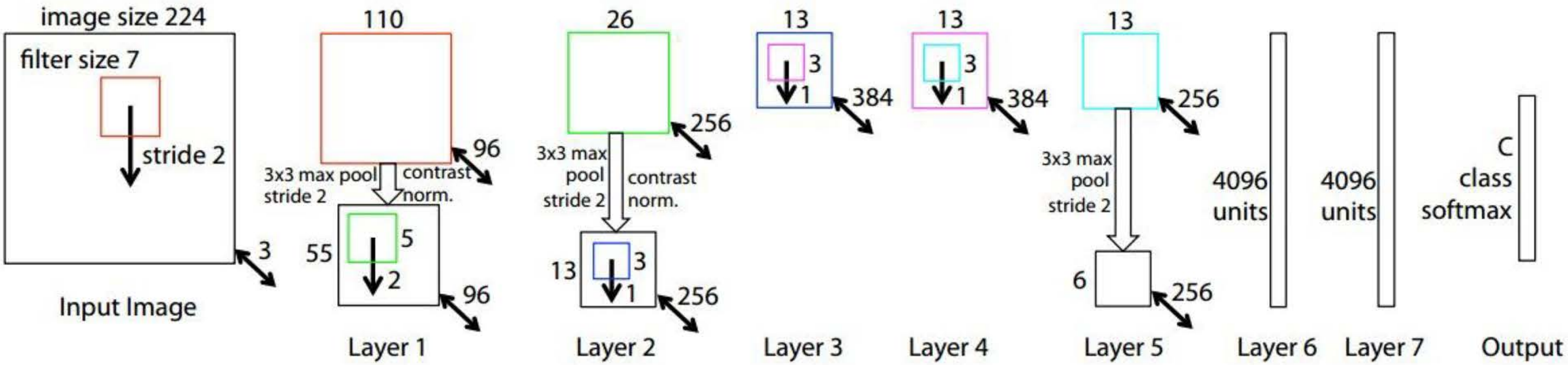
Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: ZFNet

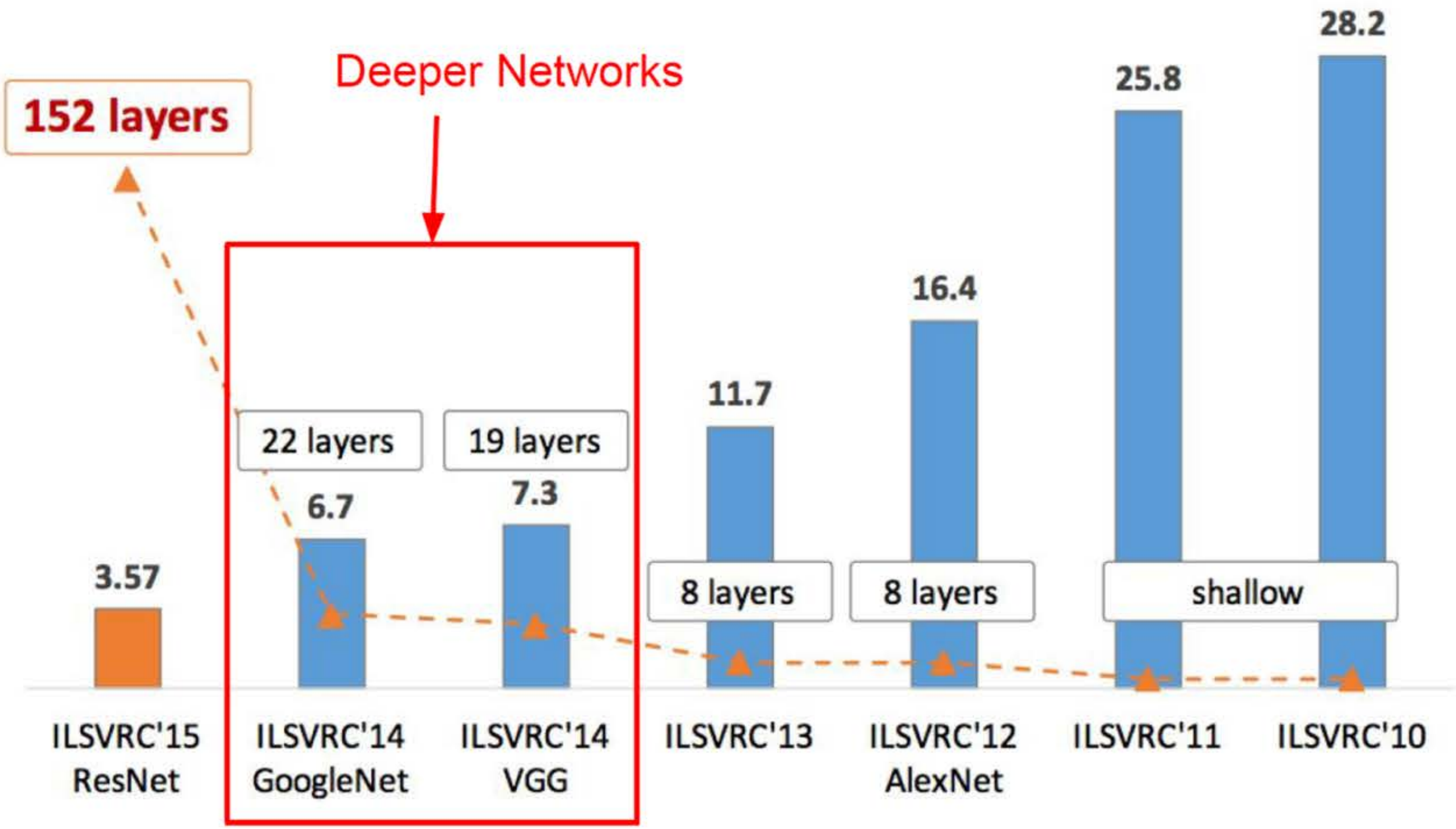
[Zeiler and Fergus, 2013]



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

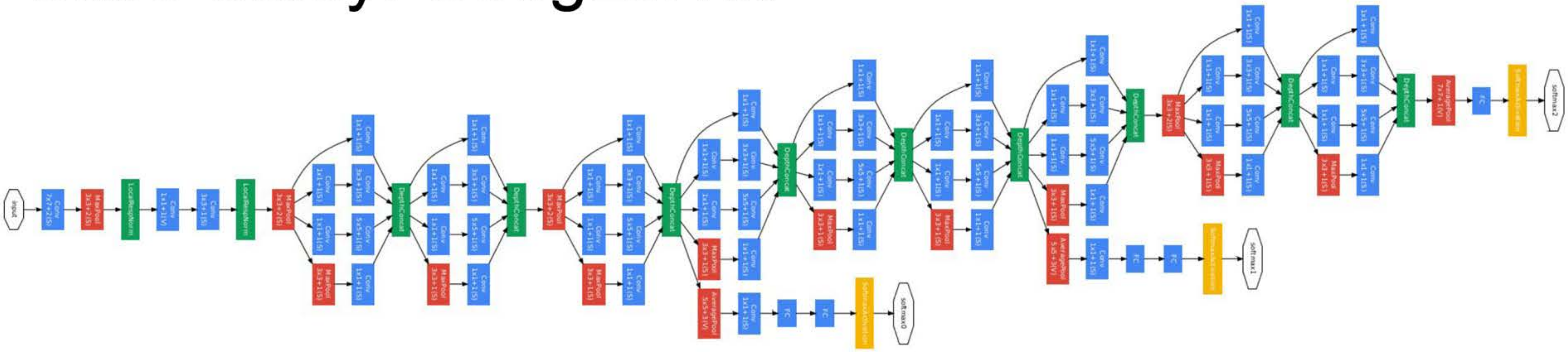
8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

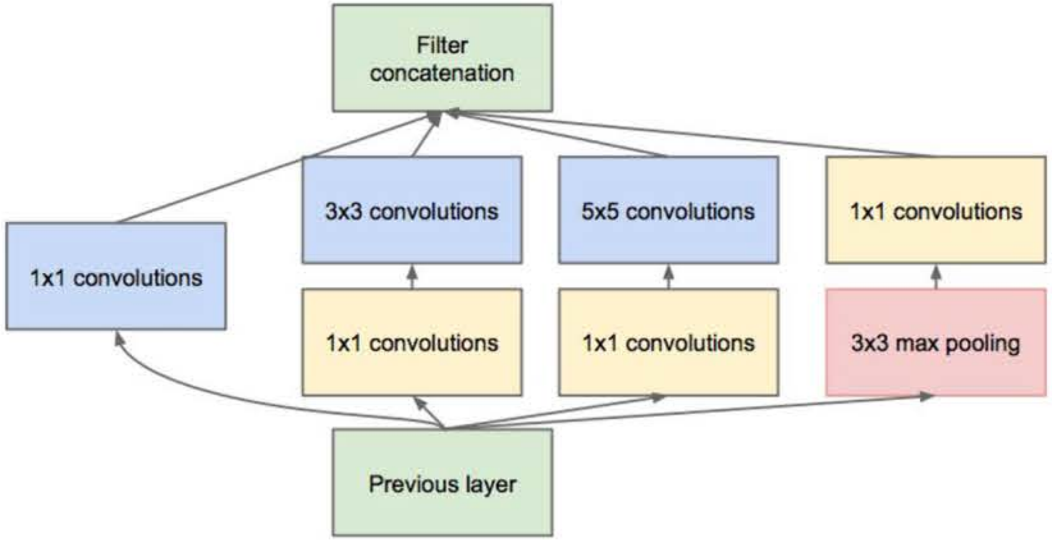
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

Case Study: GoogLeNet

[Szegedy et al., 2014]

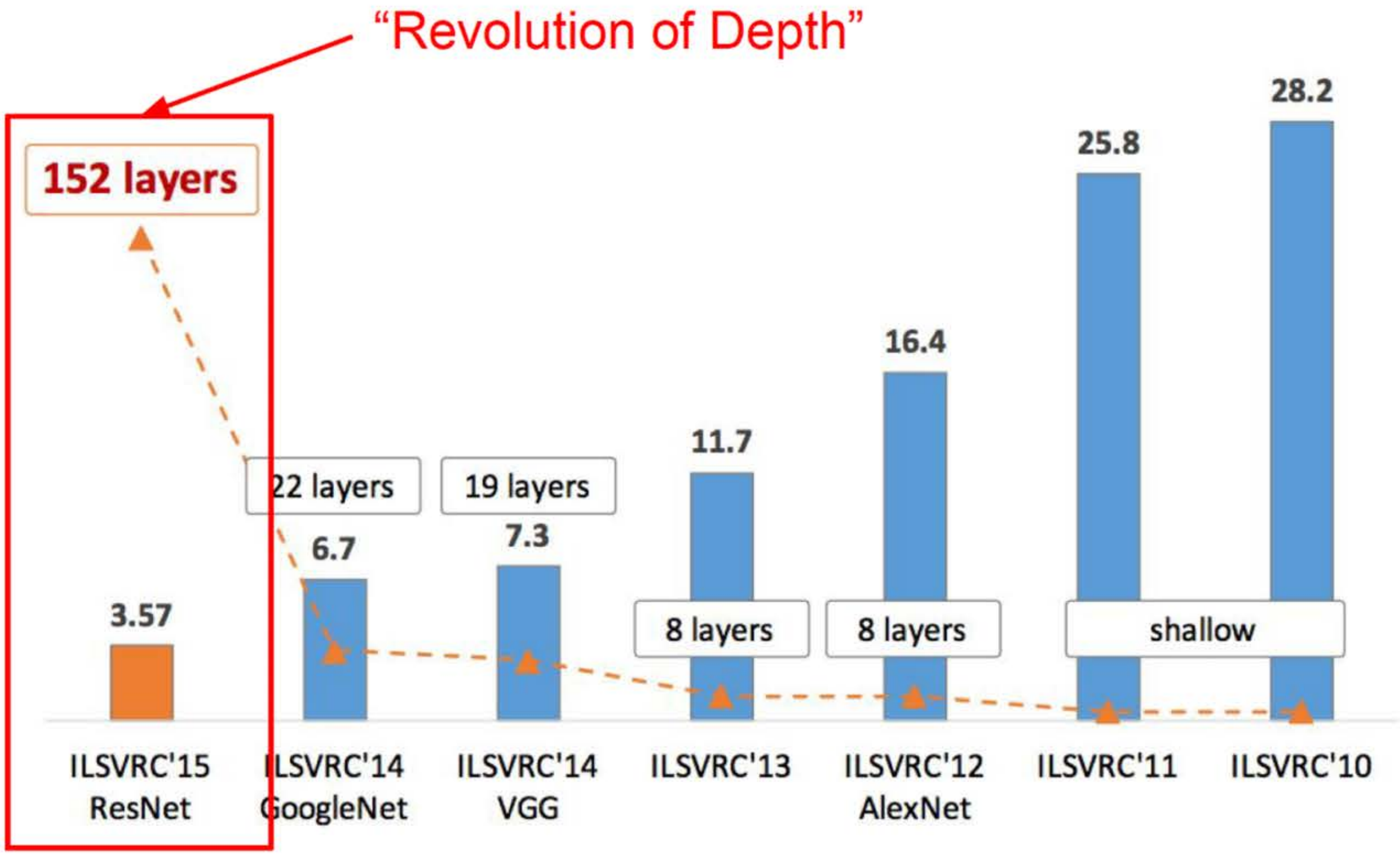




Inception module

ILSVRC 2014 winner (6.7% top 5 error)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

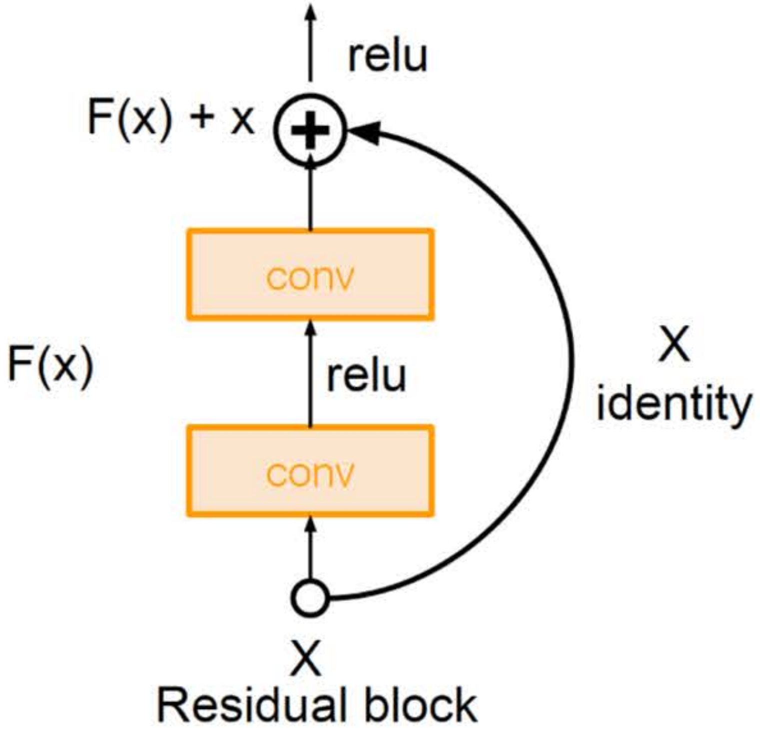


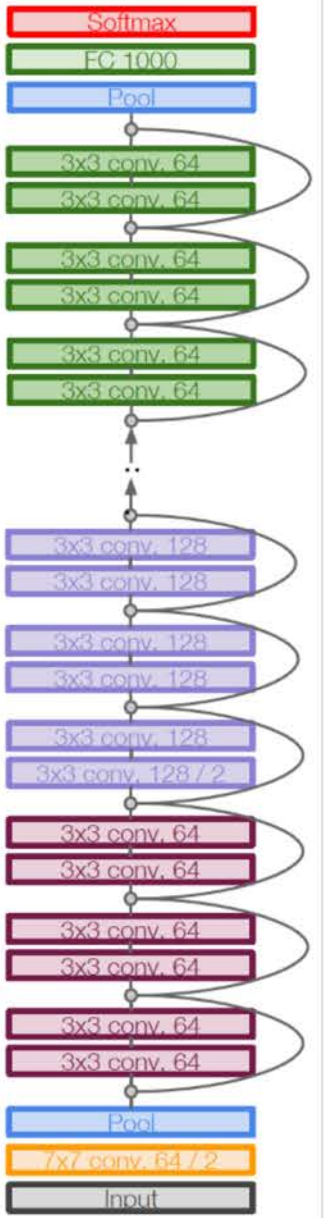
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)

Microsoft Research

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

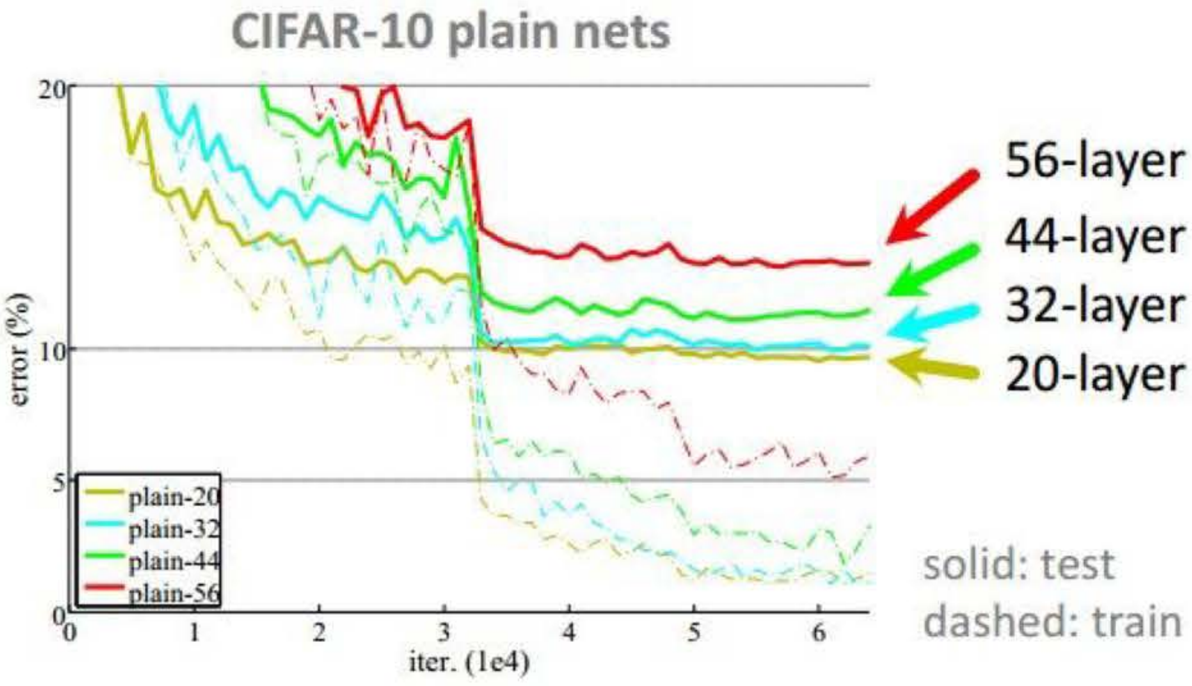
*improvements are relative numbers

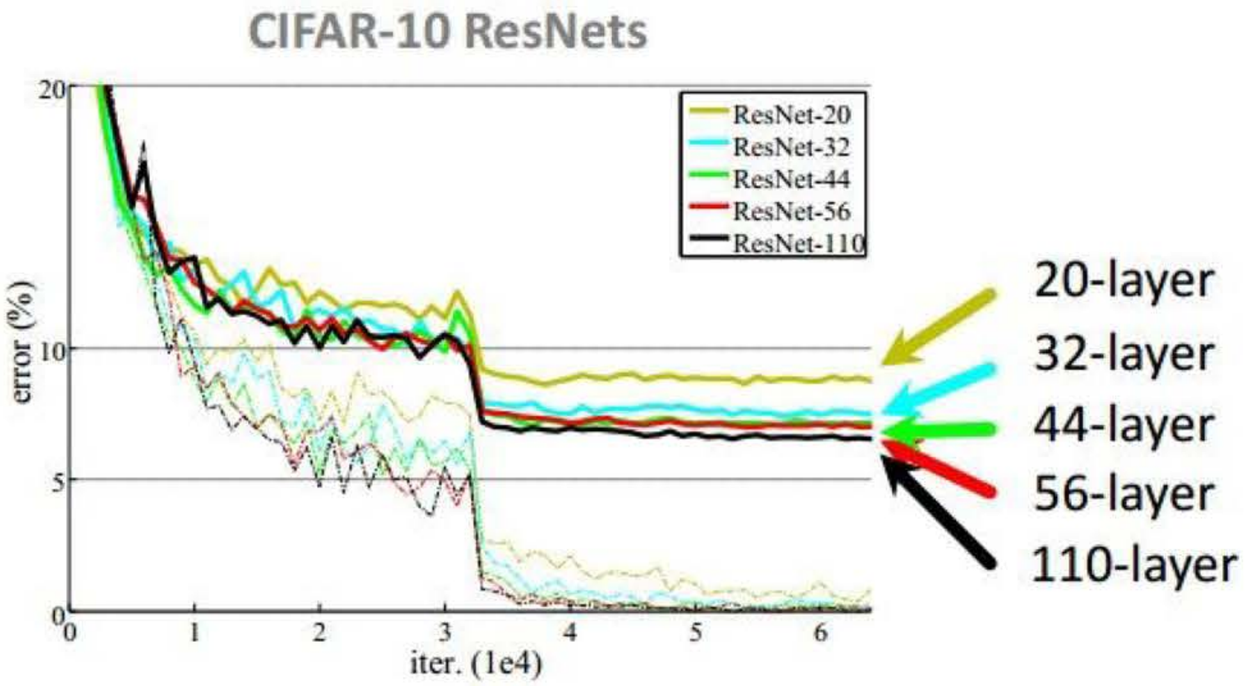
ICCV15

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

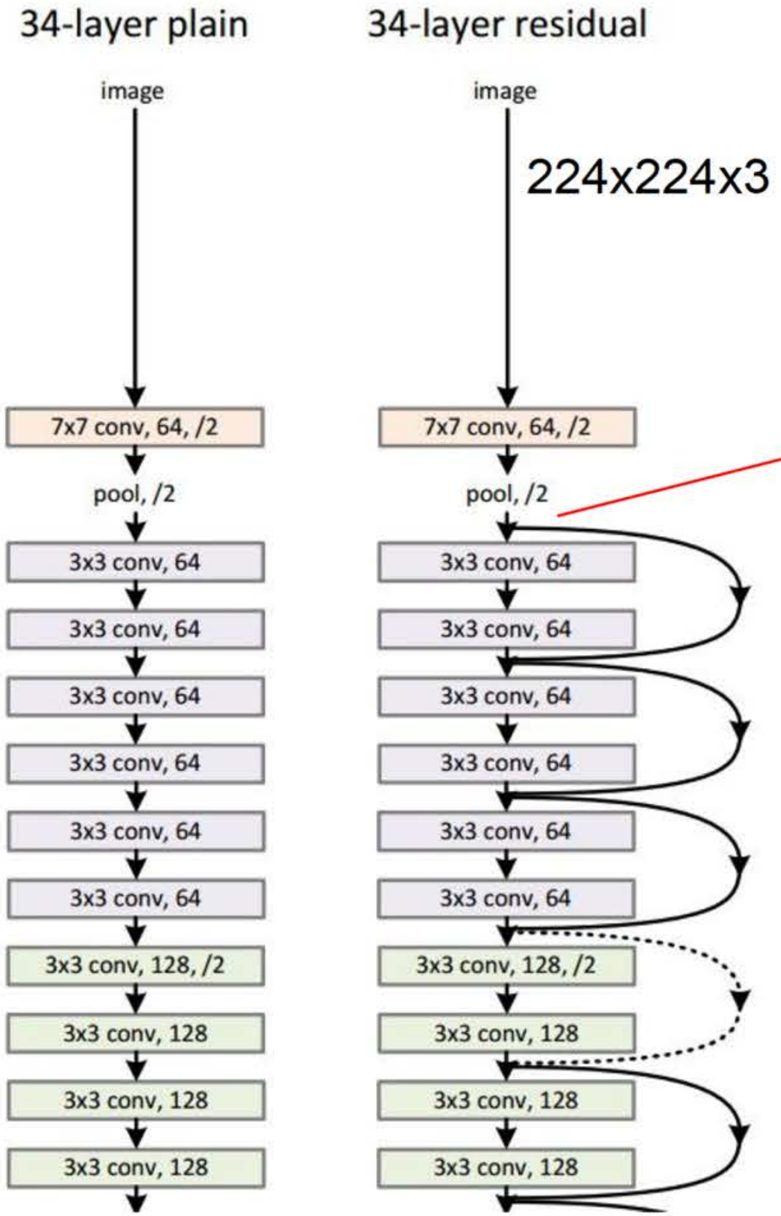
CIFAR-10 experiments





Case Study: ResNet

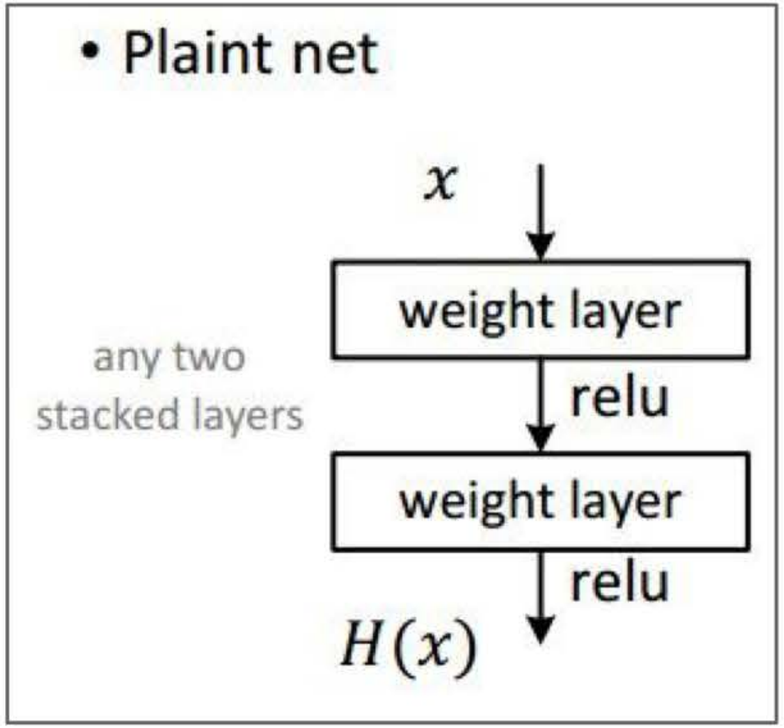
[He et al., 2015]

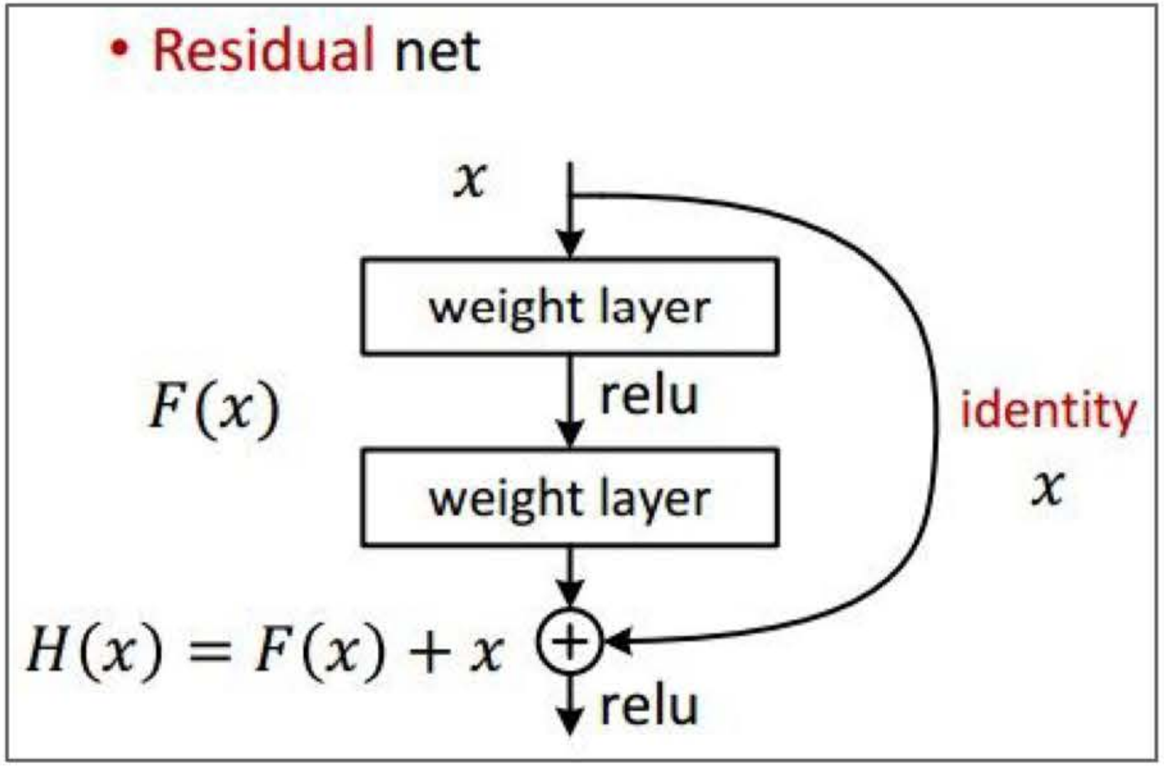


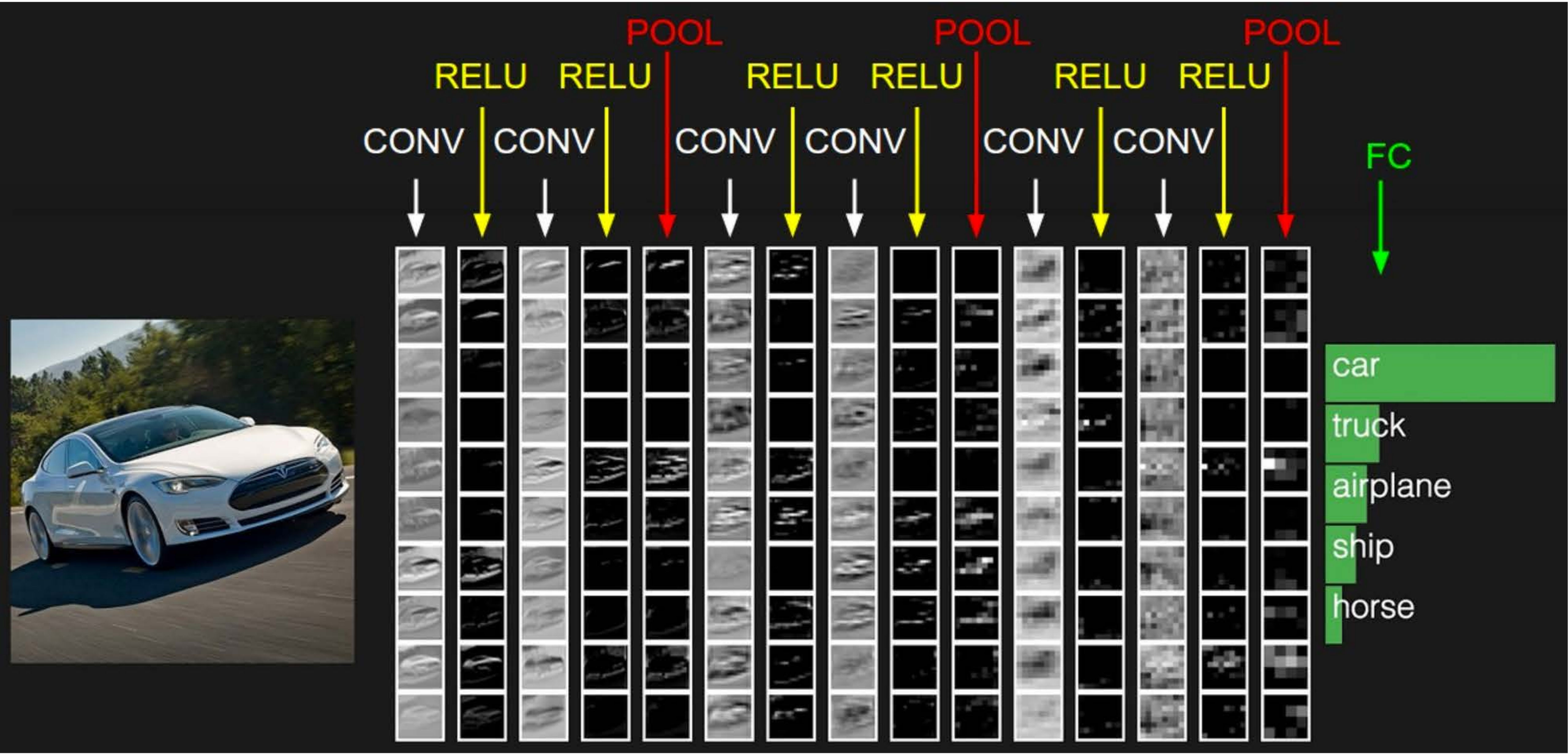
spatial dimension only 56x56!

Case Study: ResNet

[He et al., 2015]

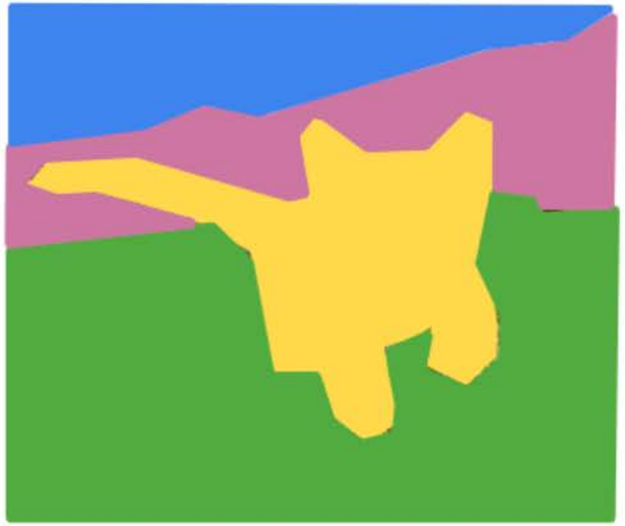




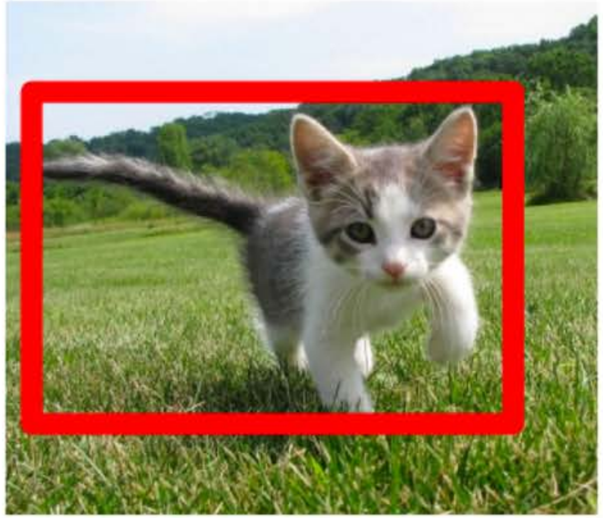

POOL
RELU
RELU
CONV
CONV
POOL
RELU
RELU
CONV
CONV
POOL
RELU
RELU
CONV
CONV
FC
car
truck
airplane
ship
horse


Sensor Fusion
2018 International
CES
TECH EVENTS
ection - NVDriveNet detection
suv-truck
car
suv-truck
suv-truck
suv-truck
suv-truck
Front:
car
car
suv-truck
suv-truck
Rear :

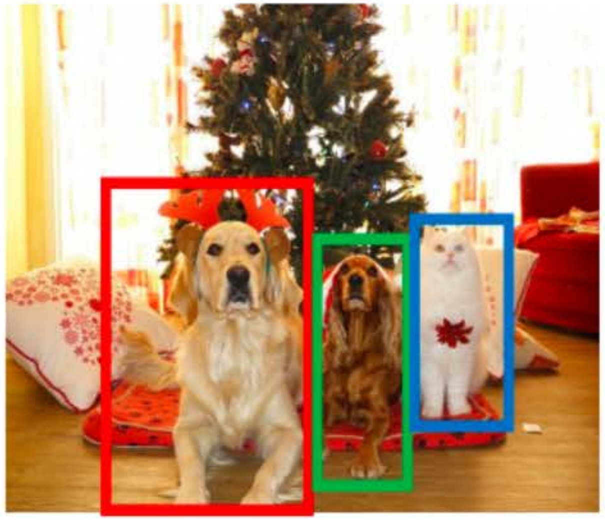
Semantic Segmentation



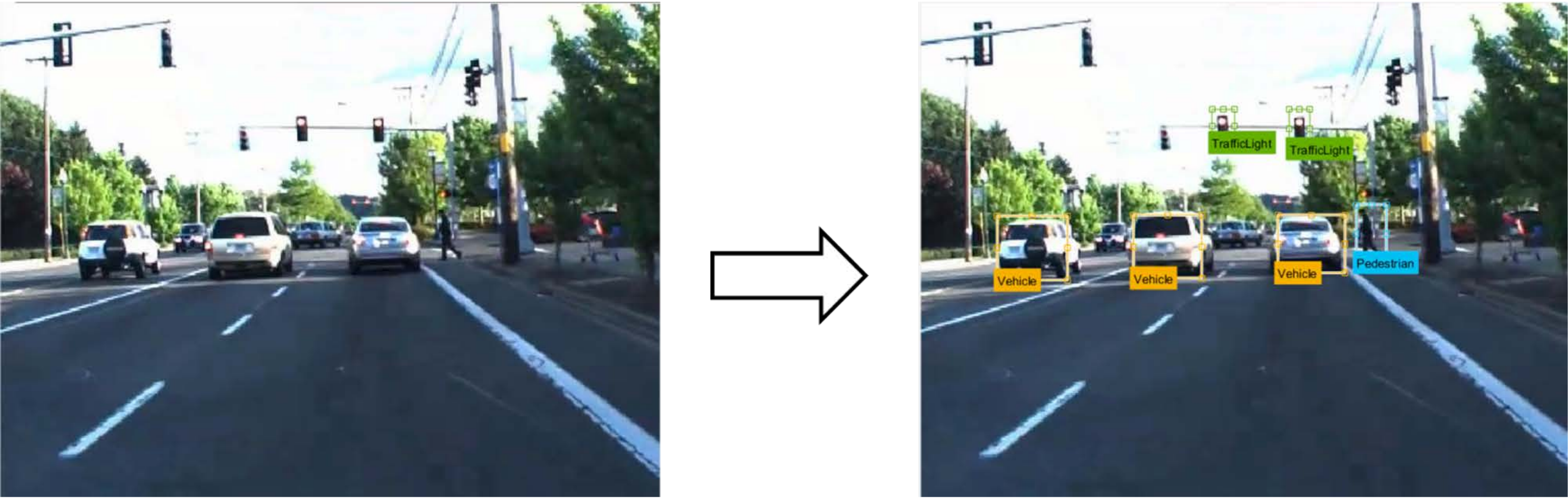
Classification + Localization



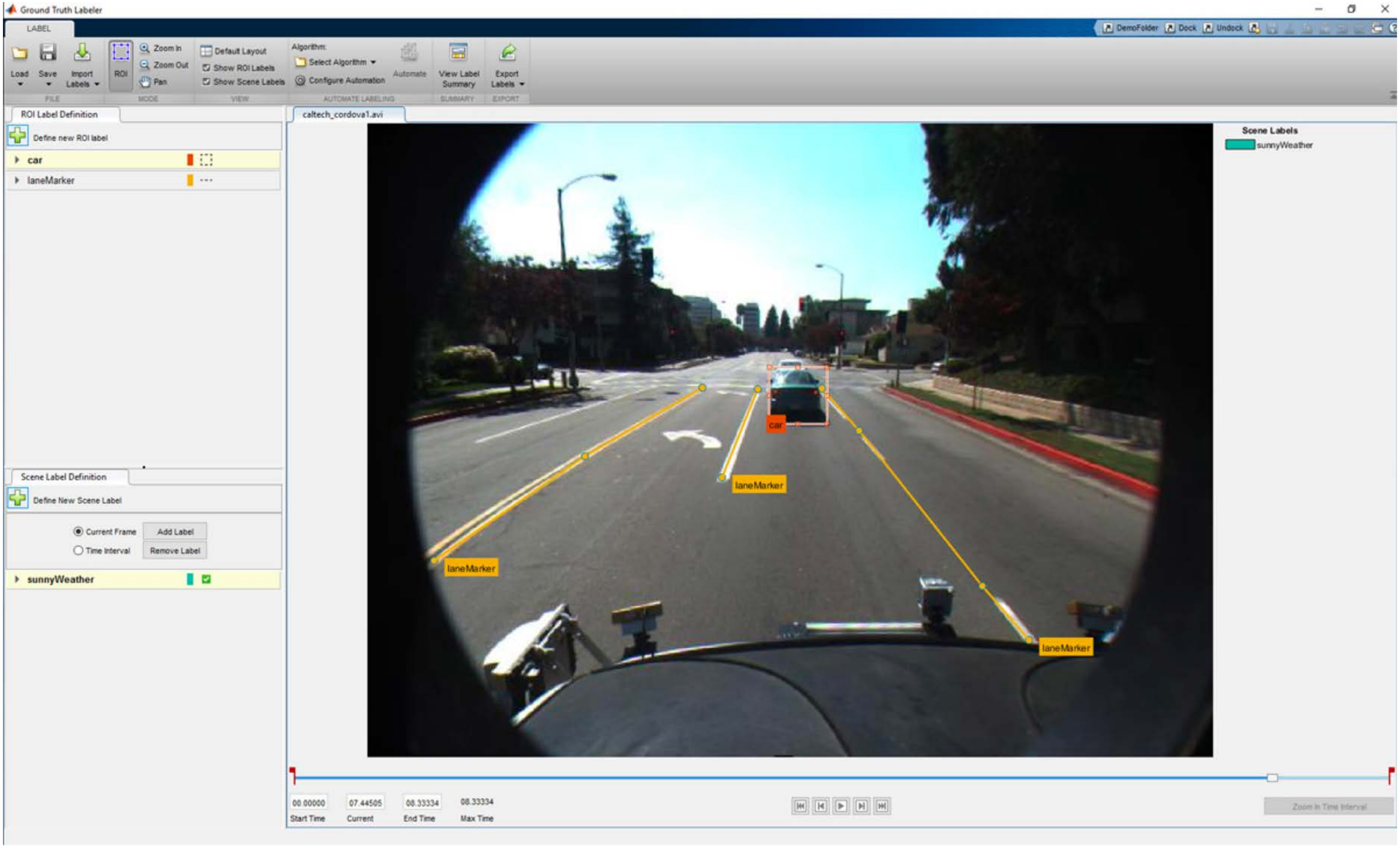
Object Detection



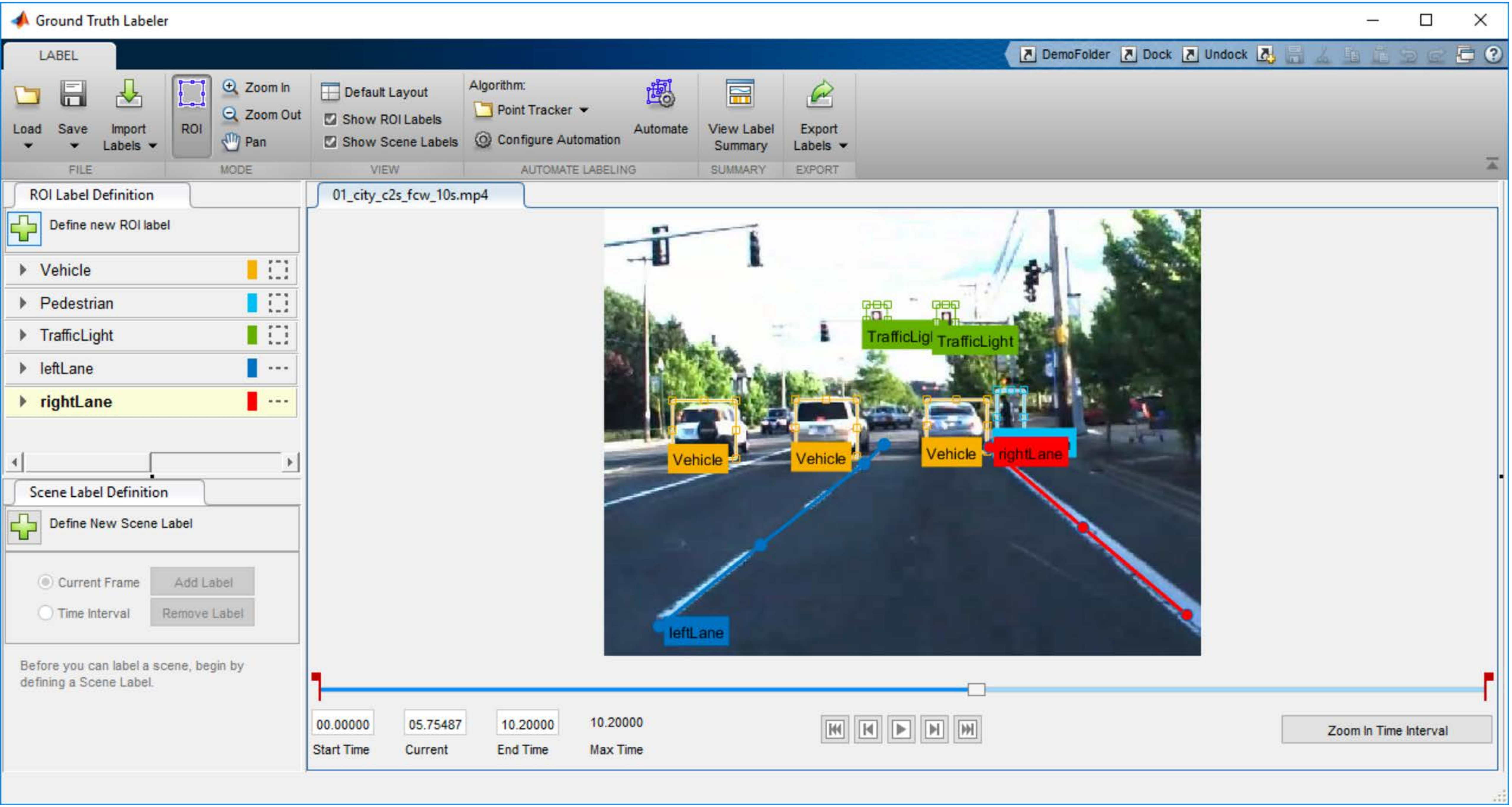
Instance Segmentation



Raw input image (left) and input image with labeled ground truth (right).


Ground Truth Labeler
LABEL
DemoFolder
Dock
Undock
Load
Save
Import Labels
ROI
Zoom In
Zoom Out
Pan
Default Layout
Show ROI Labels
Show Scene Labels
Algorithm:
Select Algorithm
Configure Automation
Automate
View Label Summary
Export Labels
FILE
MODE
VIEW
AUTOMATE LABELING
SUMMARY
EXPORT
ROI Label Definition
Define new ROI label
car
laneMarker
Scene Label Definition
Define New Scene Label
Current Frame
Add Label
Time Interval
Remove Label
sunnyWeather
caltech_cordova1.avi
Scene Labels
sunnyWeather
car
laneMarker
laneMarker
laneMarker
00.00000
Start Time
07.44505
Current
08.33334
End Time
08.33334
Max Time
Zoom In Time Interval

DATA SOURCE
Video
Image Sequence
Custom Reader
LABEL DEFINITIONS
Label Definitions
SESSION
Session
ROI Label Definition
Define New ROI Label
cars
streetLights
Scene Label Definition
Define New Scene Label
Current Frame
Add Label
Time Interval
Remove Label
Sunny
Overcast
Tunnel
gTruth
1x1 groundTruth
Property
Value
DataSource
1x1 groundTruthData...
LabelDefinitions
4x3 table
LabelData
420x4 timetable
gTruthdata.mat
LOAD
Video, Image Sequence,
or Custom Reader
DEFINE
ROIs and Scene Label Definitions
SET
Interval and Controls
LABEL
Rectangles & Lines
EXPORT LABELS
SAVE SESSION



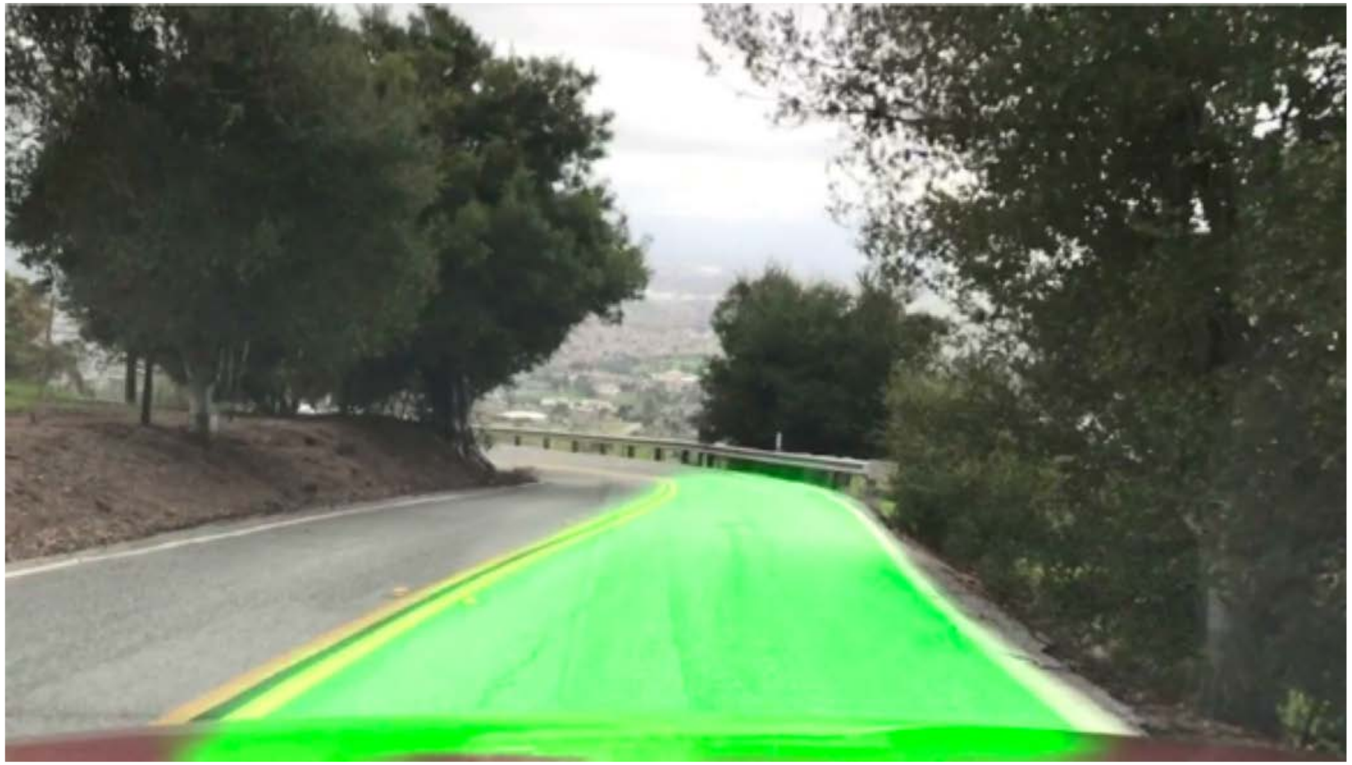
Ground Truth Labeler
LABEL
DemoFolder
Dock
Undock
Load
Save
Import Labels
ROI
Zoom In
Zoom Out
Pan
Default Layout
Show ROI Labels
Show Scene Labels
Algorithm:
Point Tracker
Configure Automation
Automate
View Label Summary
Export Labels
FILE
MODE
VIEW
AUTOMATE LABELING
SUMMARY
EXPORT
ROI Label Definition
Define new ROI label
Vehicle
Pedestrian
TrafficLight
leftLane
rightLane
Scene Label Definition
Define New Scene Label
Current Frame
Add Label
Time Interval
Remove Label
Before you can label a scene, begin by defining a Scene Label.
01_city_c2s_fcw_10s.mp4
TrafficLight
Vehicle
Vehicle
Vehicle
rightLane
leftLane
00.00000
Start Time
05.75487
Current
10.20000
End Time
10.20000
Max Time
Zoom In Time Interval

regressionOutputs =

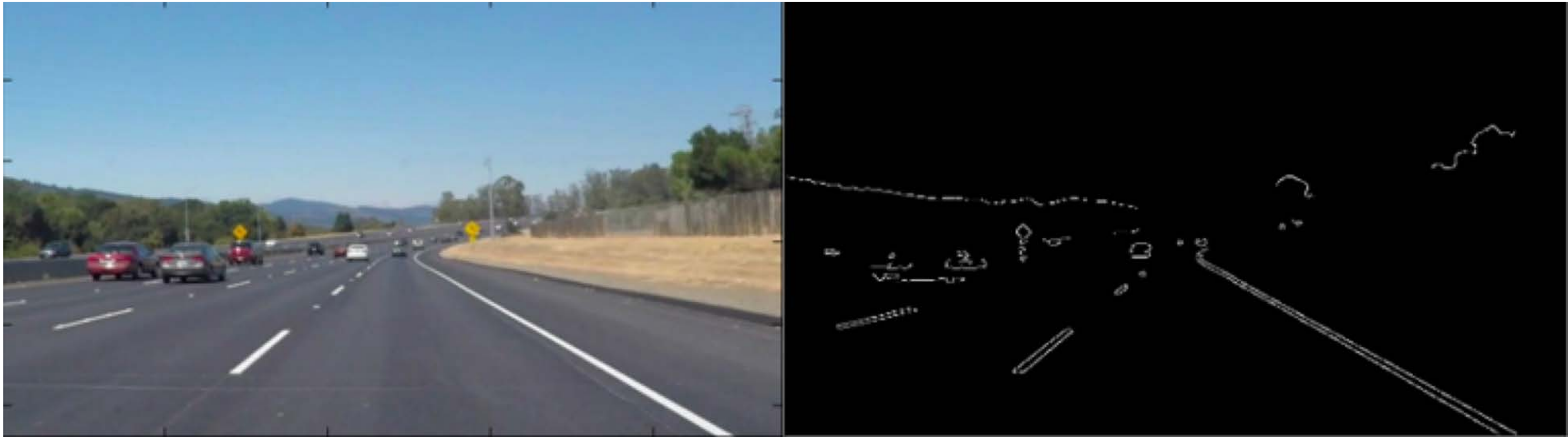
1225×6 table

leftLane_a	leftLane_b	leftLane_c	rightLane_a	rightLane_b	rightLane_c
3.5482e-05	0.0060327	1.7599	-0.00015691	0.030256	-2.0559
-3.9519e-05	0.014116	1.662	-0.00097636	0.02979	-2.0749
-6.778e-07	-0.00063158	1.776	-7.0963e-05	0.0024721	-1.9428
-0.00023646	0.0088324	1.8188	-0.00050391	-0.0015166	-1.973
-0.00055867	0.012996	1.8074	-8.6643e-05	0.00098652	-1.935

Lane Detection with Deep Learning

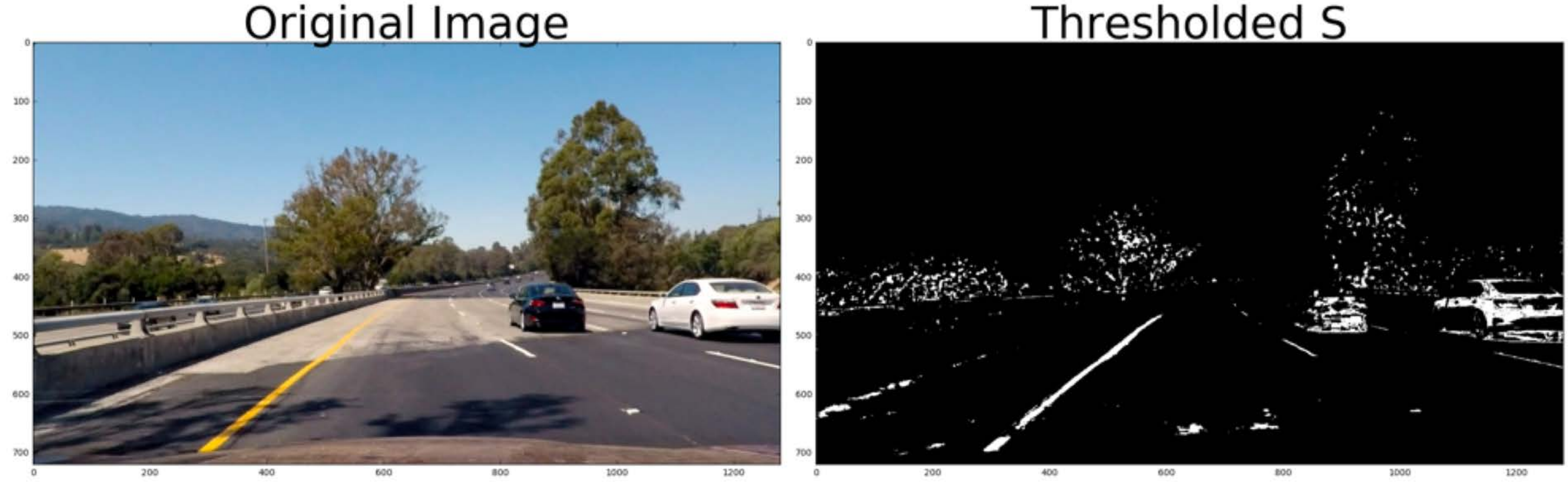


Canny Edge Detection

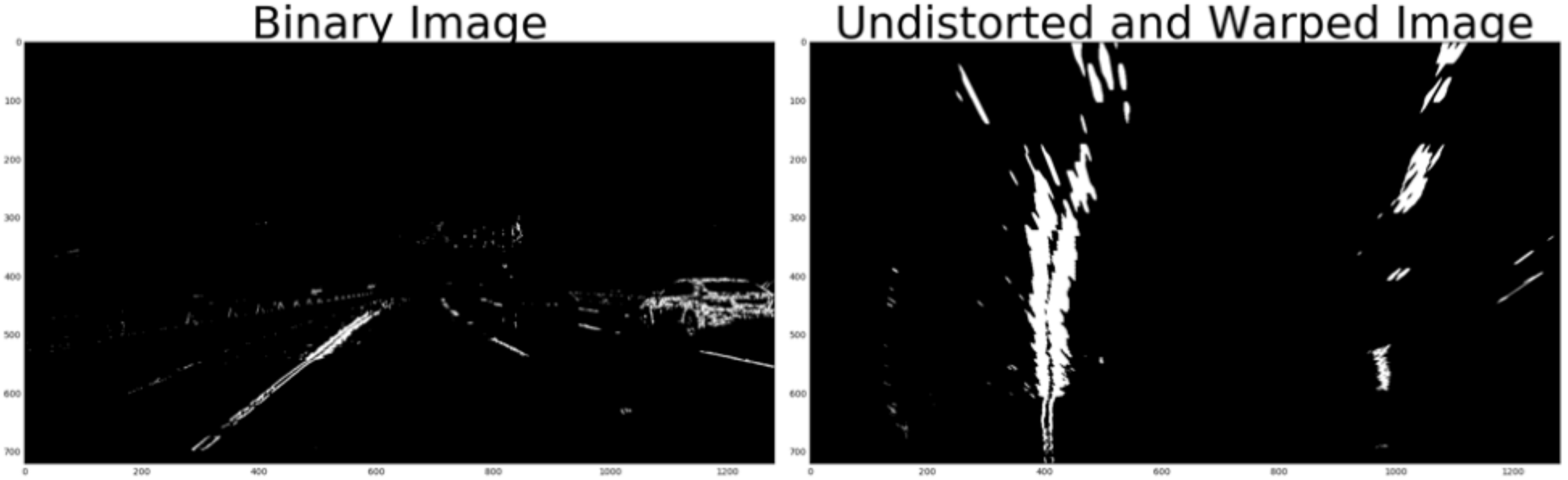


Perspective Transformation of an Image

The 'S' channel, or Saturation, with binary activation



A few more thresholds (left) for activation, with the resulting perspective transformation

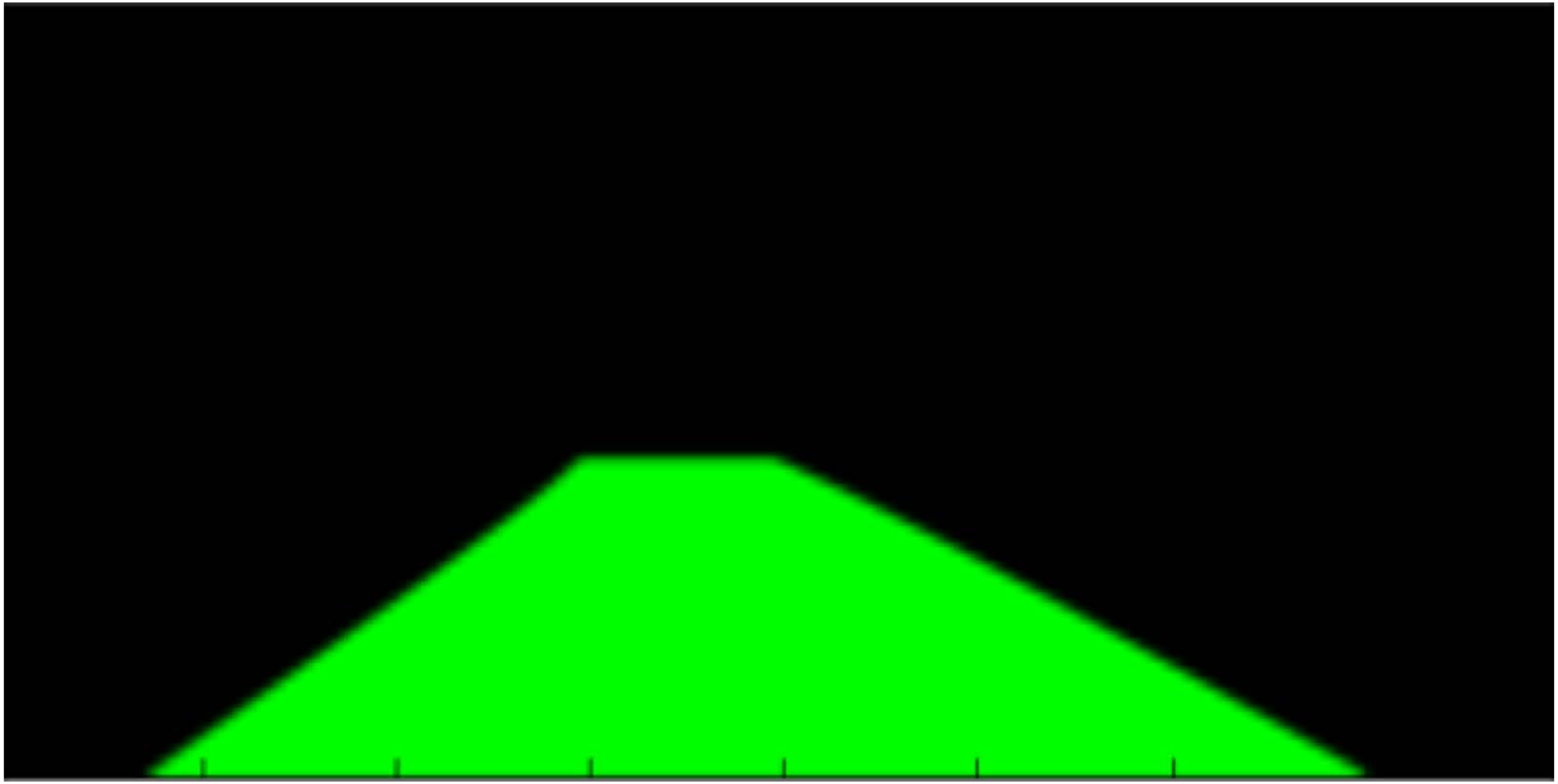


Sliding windows and a decent-looking result

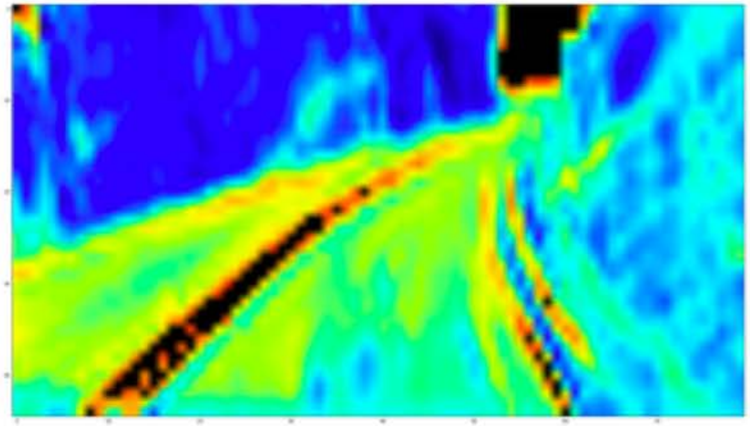
- Perspective transformation is fairly specific to the camera
- Gradient and color thresholds only work in a small set of conditions
- Slow 5-8 fps

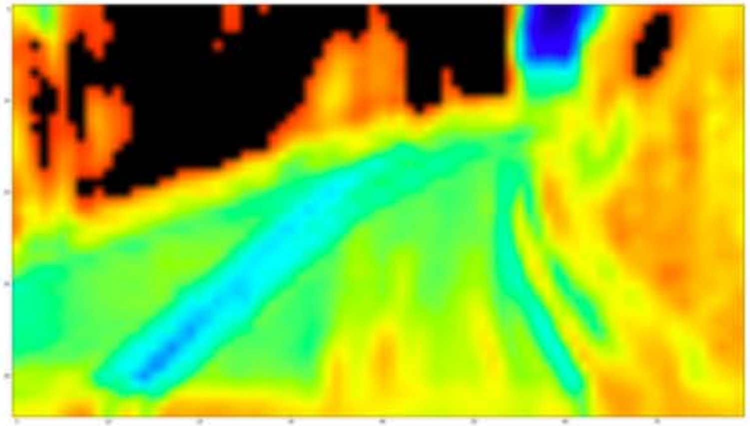

0.27 meters left of center
Radius of Curvature = 3.0(m)

One of the new labels — a lane image



Activation maps of the first few layers





Top left: Input – Perspective Transformed Image
Output – Six polynomial coefficients

Top right: Input – Road Image
Output – Six polynomial coefficients

Bottom left: Input – Road Image
Output – Lane in 'G' color channel


HG05WWF
© BNPS.CO.UK