Bauhaus-Universität Weimar

Image Analysis and Object Recognition

Assignment 6

Deep Convolutional Neural Networks for Scene Classification

SS 2017

(Course notes for internal use only!)

- Assignment dates:
  - 13.04.2017 → Introduction + Assignment 1
  - 04.05.2017 → Assignment 1+2
  - 18.05.2017 → Assignment 2+3
  - 01.06.2017 → Assignment 3+4
  - 15.06.2017 → Assignment 4+5
  - **29.06.2017 → Assignment 5+6**
  - 13.07.2017 → Final Meeting / Summary / Discussion

A4 Comments

$$log\left(\alpha_c \cdot p\left(x_i|\mu_c^k, \Sigma_c^k\right)\right) = log(\alpha_c) - \frac{1}{2}\left[log\left(\left|\Sigma_c^k\right|\right) + \left(x_i - \mu_c^k\right)^T {\Sigma_c}^{-1}\left(x_i - \mu_c^k\right)\right]$$

```
% logarithmic probability of all vectors for all components
function LnVectorProb = CalcLnVectorProb(model, trainVect)

    N_c = size(model.weight)(1);
    N_x = size(trainVect)(1);
    LnVectorProb = zeros(N_c, N_x);

    for c=1: N_c
      covar = squeeze(model.covar(c,:,:));
      covar_det = det(covar);
      covar_inv = inv(covar);
      for x=1: N_x
        shiftedTrain = trainVect(x,:) - model.mean(c,:);
        LnVectorProb(c, x) = log(model.weight(c)) - 0.5 * (log(covar_det) + shiftedTrain * covar_inv * shiftedTrain');
      end
    end

end
```

A4 Comments

- E-Step with loops:

```
function LnCompProb = GmmEStep(model, trainVect)
    num_mod = length(model.weight);
    [num_vec,~] = size(trainVect);
    nominator = CalcLnVectorProb(model, trainVect);
    LnCompProb = zeros(num_mod, num_vec);

    for c = 1: num_mod
        for j = 1: num_vec
            denominator = log( sum( exp(nominator(:,j)) ));
            LnCompProb(c,j) = nominator(c,j) - denominator;
        end
    end
end
```

- E-Step **without** loops:

```
function LnCompProb = GmmEStep(model, trainVect)

    N_c = size(model.weight)(1);
    N_x = size(trainVect)(1);

    LnVectorProb = CalcLnVectorProb(model, trainVect);

    denom = exp(LnVectorProb);
    denom = log(repmat(sum(denom), N_c, 1));

    LnCompProb = LnVectorProb - denom;
end
```

A4 Comments

- E-Step denominator:
  - Aim: calculate $\log(a+b)$ given $\log(a)$ and $\log(b)$
  - Our solution: $\log(a+b) = \log(\exp(a) + \exp(b))$
    - → **May cause numerical problems!**
- Better solution: introduce a scale factor s for computations

$$\log\left(\sum_{i=1}^{N} z_i\right) = \log\left(s \cdot \sum_{i=1}^{N} \frac{1}{s} z_i\right)$$

$$= \log\left(s \cdot \sum_{i=1}^{N} \exp(\log(z_i) - \log(s))\right)$$

$$= \log(s) + \log\left(\sum_{i=1}^{N} \exp(\log(z_i) - \log(s))\right)$$

- Suitable choice for s: $\log(s) = \max(\log(z_i), i = 1, \dots, N)$

A4 Comments

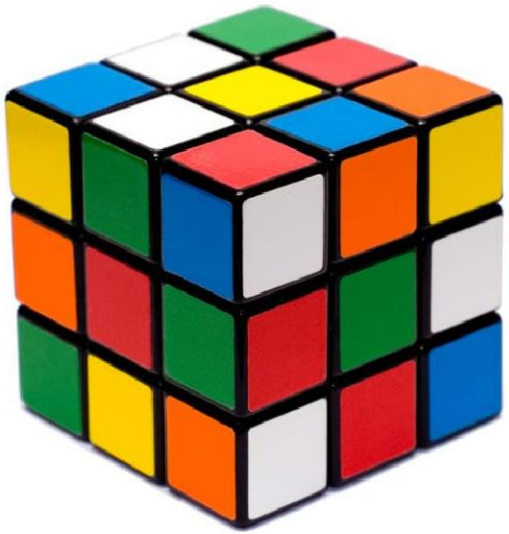
```
function LnCompProb = GmmEStep(model, trainVect)

    % current number of components
    n_comp = numel(model.weight);

    % logarithmic probability of all vectors for all components
    % this is the enumerator of p(y=c|x,omega)
    LnVectorProb = CalcLnVectorProb(model, trainVect);

    % the above approach of normalization can lead to numerical problems
    % do normalization here!
    % 1) get the maximum probability for each feature vector
    max_LnVectorProb = max(LnVectorProb,[],1);
    % 2) resize array according to number of components
    scaling_factors = repmat(max_LnVectorProb, n_comp, 1);
    % 3) subtract scaling_factors from LnVectorProb
    % 4) take exp of the result
    % 5) summarize the n_comp values for each feature vector
    % 6) take the logarithm of the sums
    % 7) add the maximum to the result of 6 --> scaling denominator for the
    %    probabilities in LnVectorProb
    denominator = max_LnVectorProb + log(sum(exp(LnVectorProb - scaling_factors),1));

    % the result "denominator" is a vector --> reshape to apply the
    % division to all feature vector values
    denominator = repmat(denominator, n_comp, 1);

    % computatiion of the wanted probabilities LnCompProb
    % --> division of numbers is equal to difference of their log values
    LnCompProb = LnVectorProb - denominator;
end
```

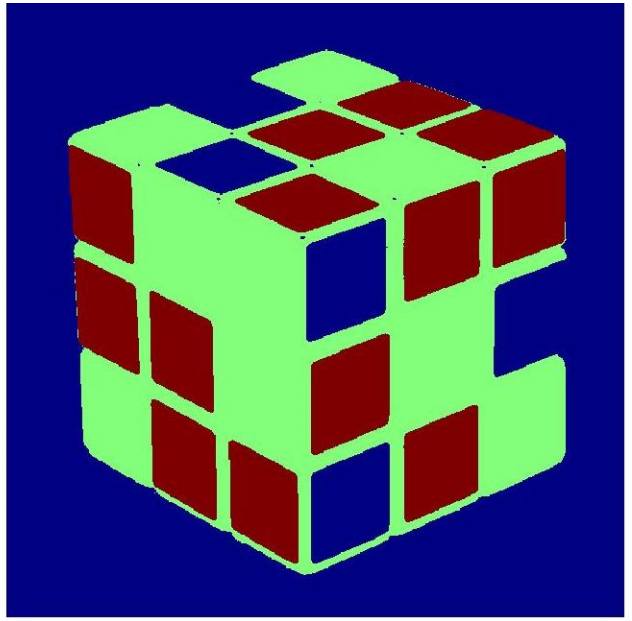
$$\log(s) = \max(\log(z_i), i = 1, \dots, N)$$

$$Denom = \log(s) + \log\left(\sum_{i=1}^{N} \exp(\log(z_i) - \log(s))\right)$$

$$log\left(p\left(y_i = c|x_i, \Omega_c^k\right)\right) = Num_{c,i} - Denom_i$$

Results with 3 Clusters

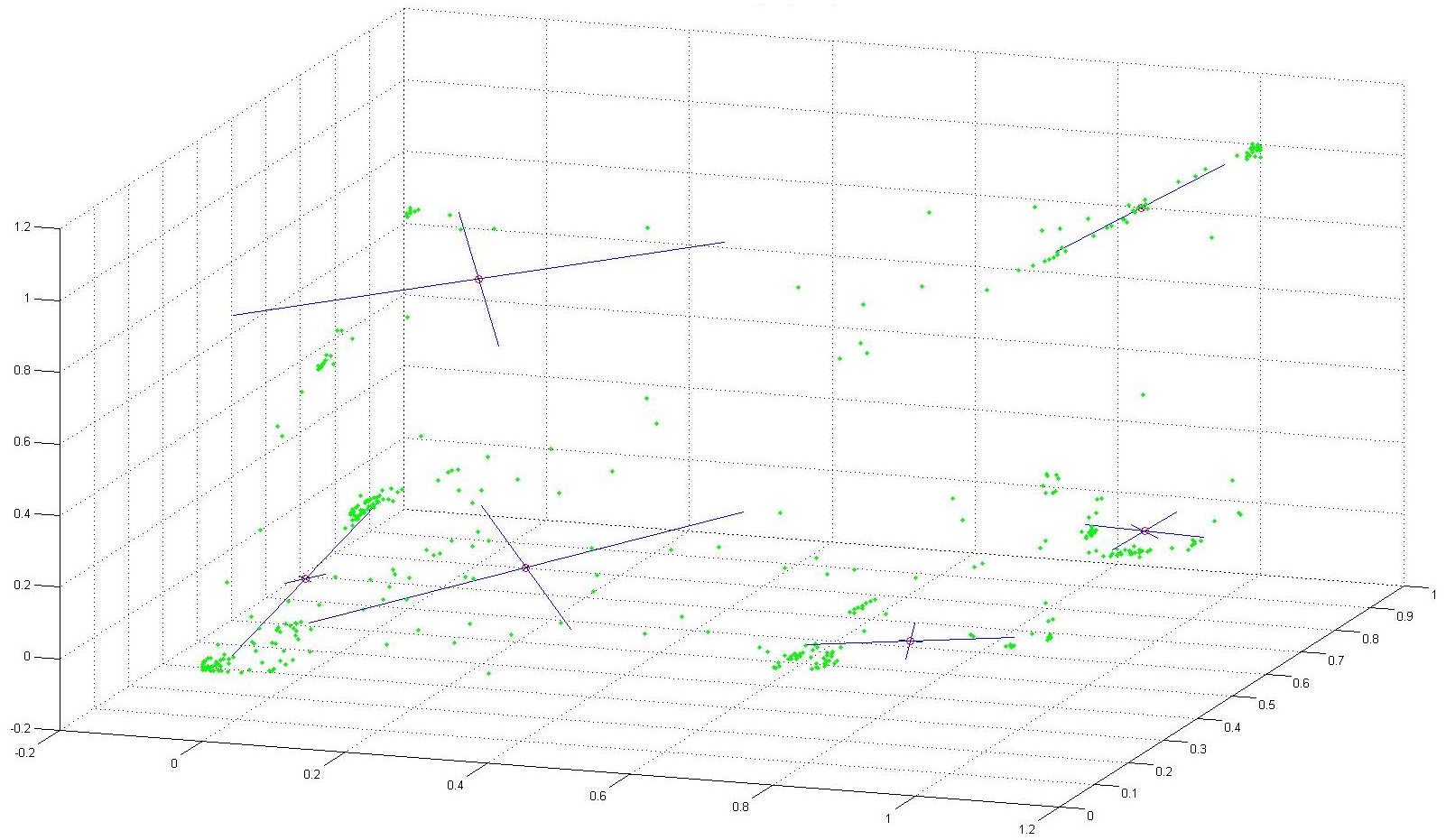




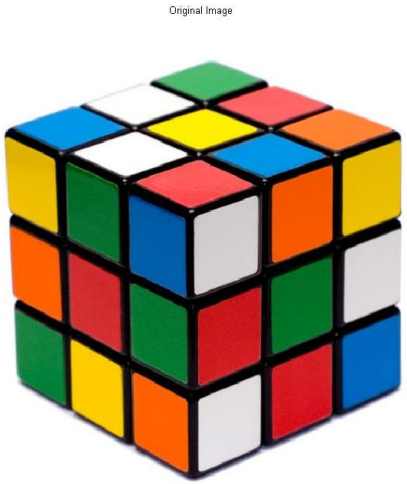
- Not all patches/colors separated

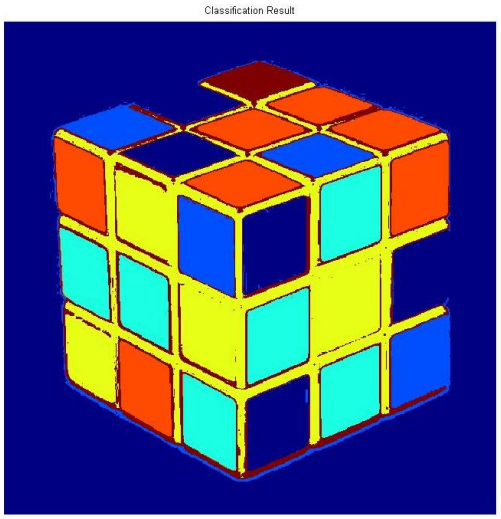
Results with 6 Clusters

- Feature Space



Results with 6 Clusters

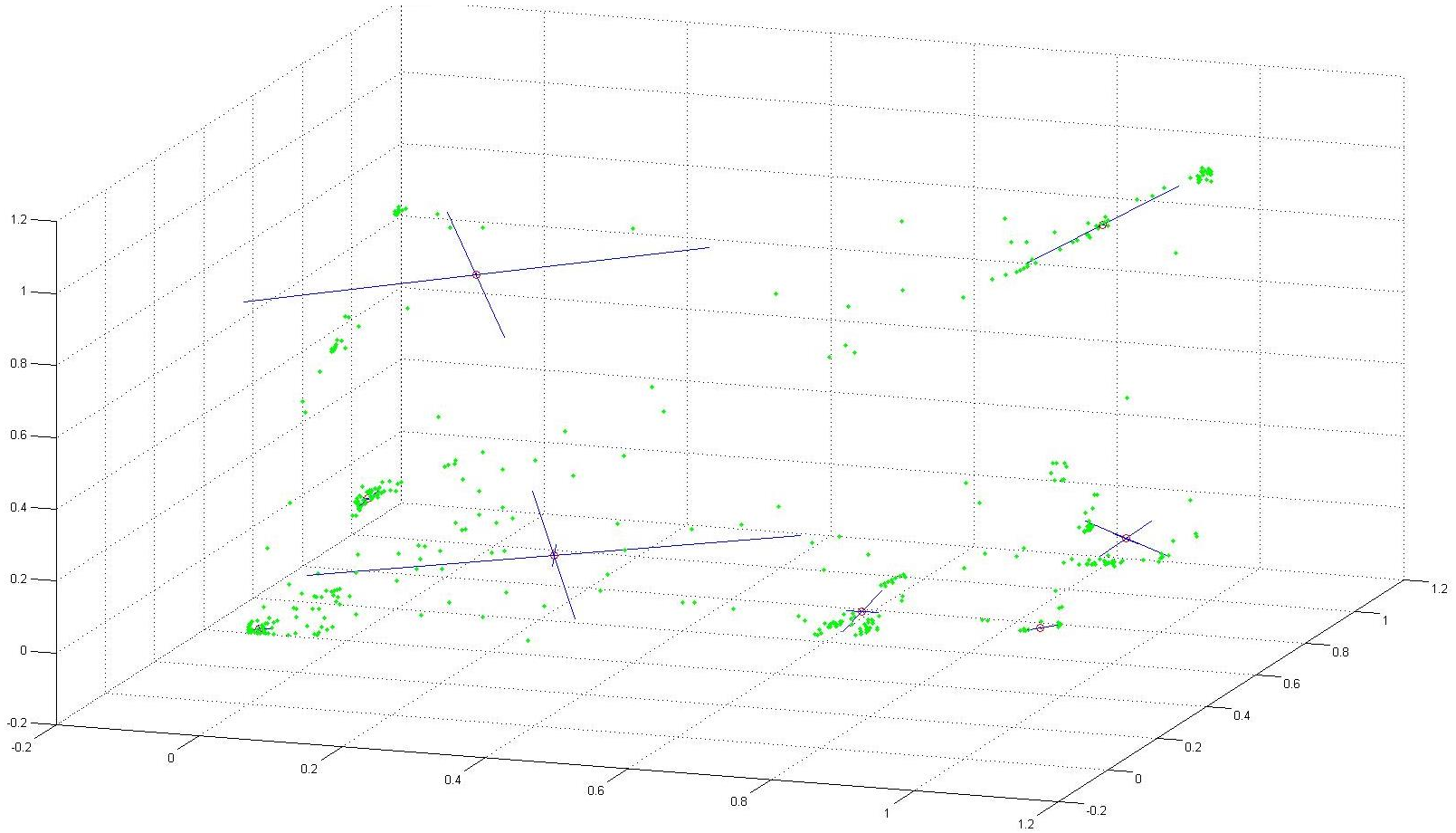




- Patches and lines between them not well seperated
  - Confusions between similar colors, e.g. yellow/orange/red
  - Confusions due to illumination changes (e.g. green)
  - 6 Colors + Background + Black lines → more than 6 clusters needed for classification

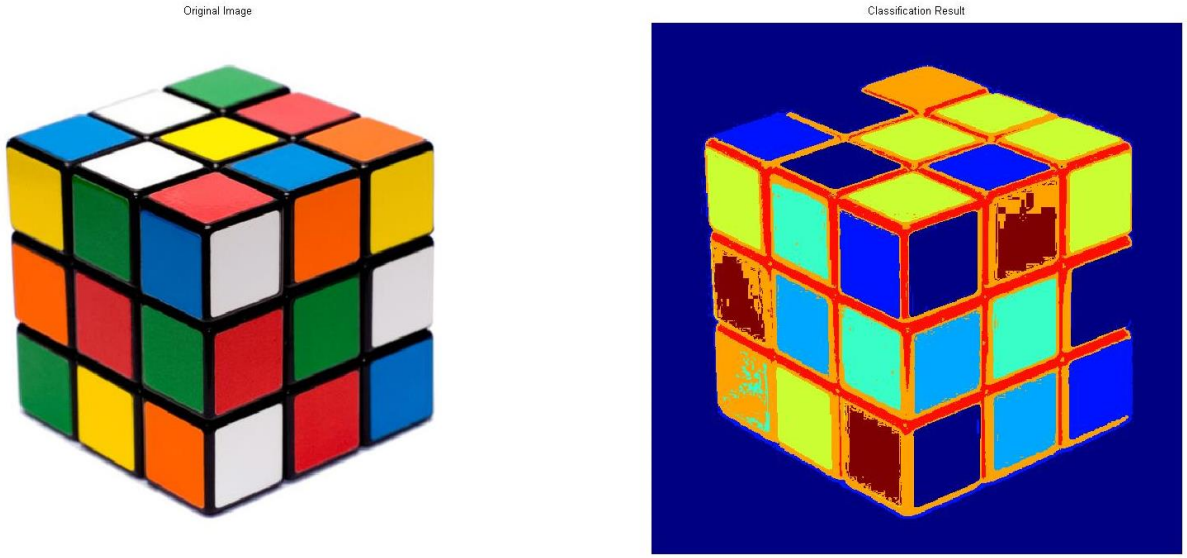
Results with 8 Clusters

- Feature Space



Results with 8 Clusters

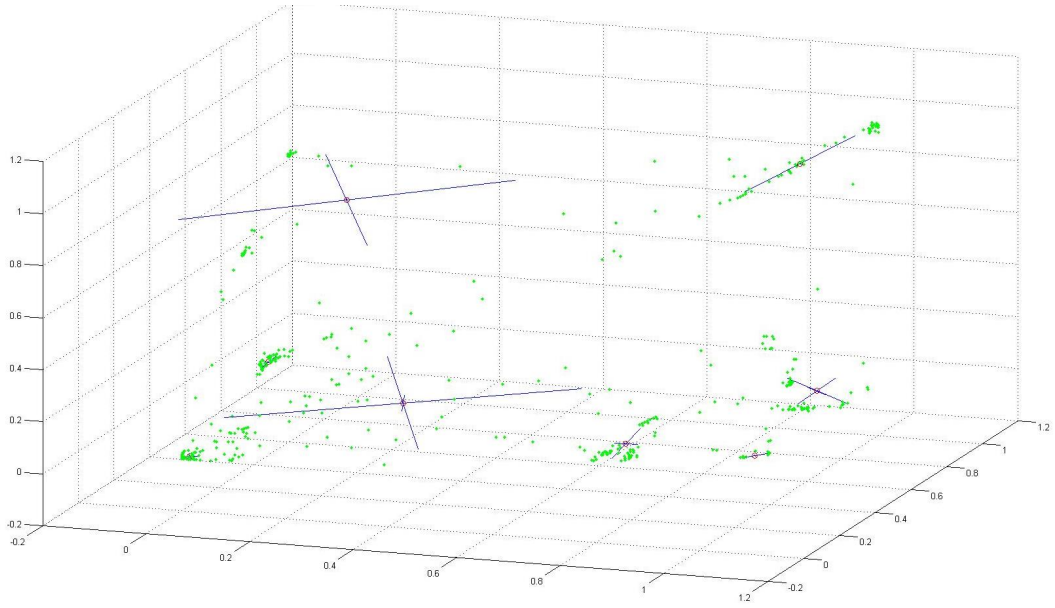
- Segmentation result



- Border areas (black in image) appear as red in result
- Some confusions remain!
- Possible solution to achieve better results: Use other **criterion to split** clusters

Splitting of Clusters

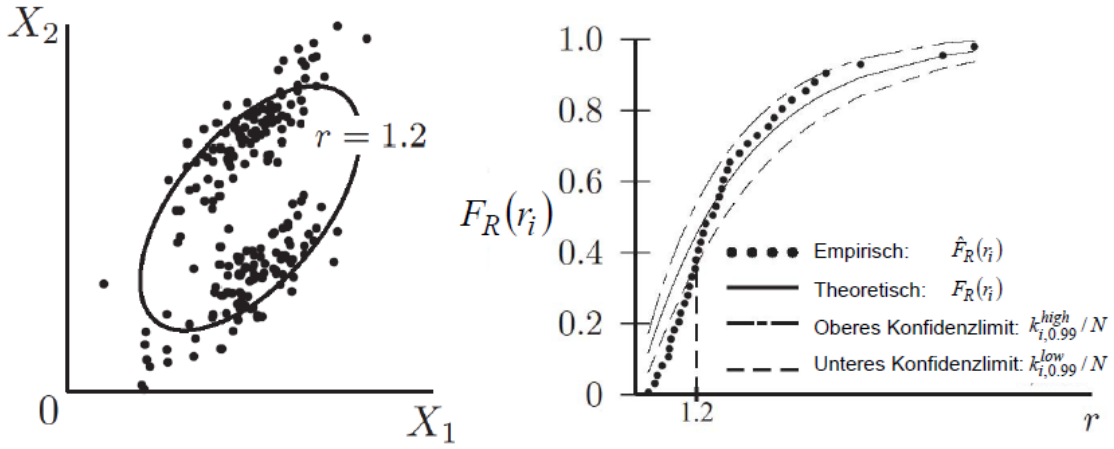
- Here: Splitting largest cluster



- Other strategies:
  - Maximum standard deviation
  - Distribution of feature vectors
  - …

Splitting of Clusters

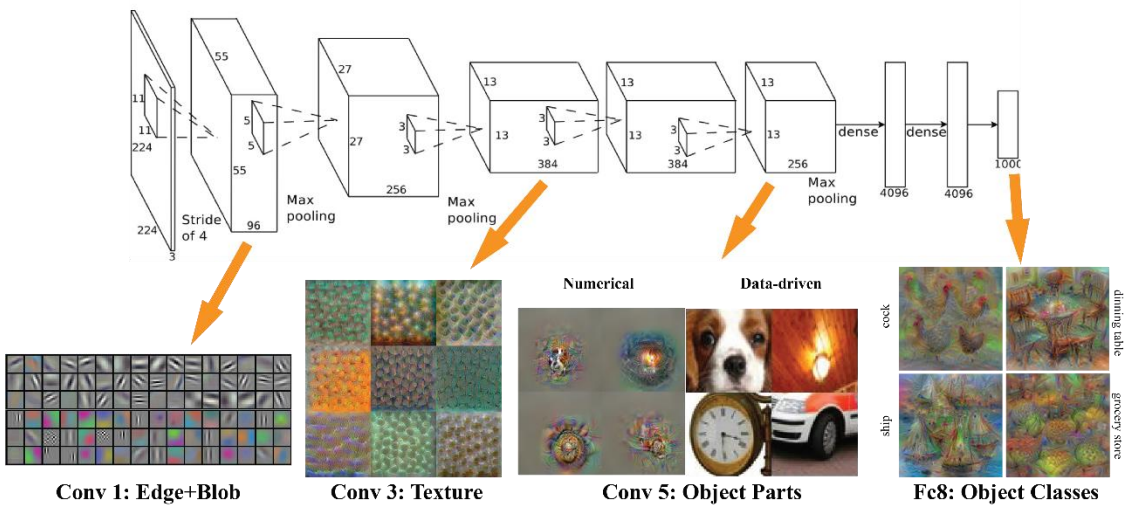
- Multivariate test: Analyzing the distances to cluster centroid



- r → Mahalanobis distances
- $F_R(r_i)$ → Theoretical cumulative distribution function
- $F_R(r_i)$ → Empirical cumulative distribution function

Assignment 6

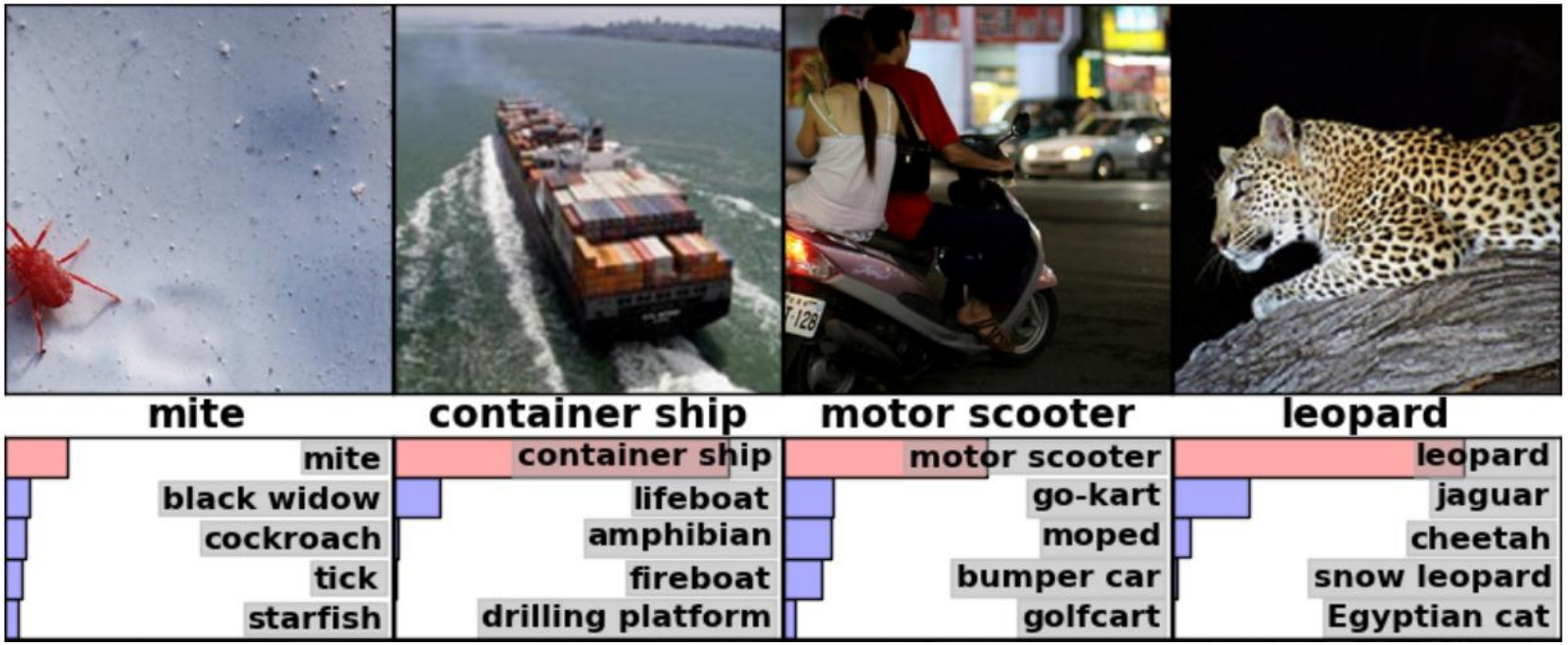
Deep Convolutional Neural Networks for Scene Classification



http://vision03.csail.mit.edu/cnn_art/index.html

Applications of CNNs

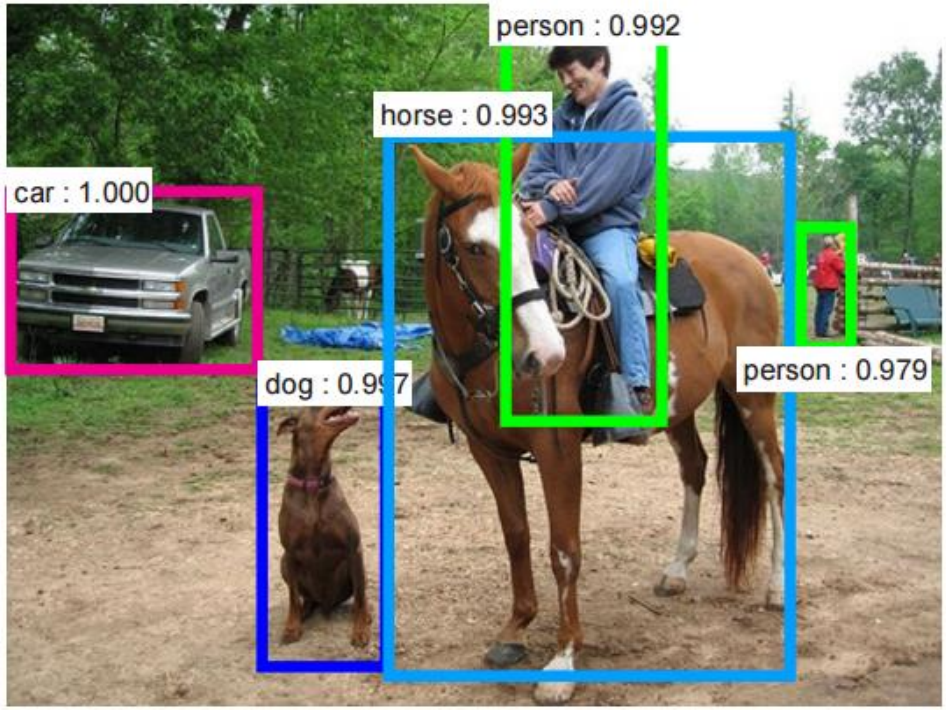
Image Classification

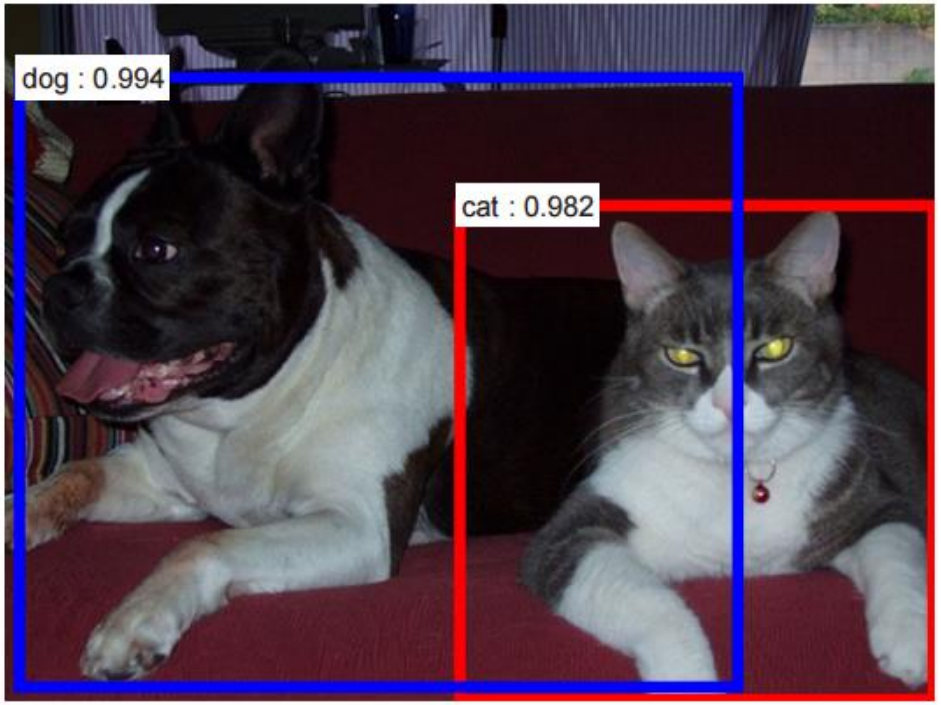


Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.

Applications of CNNs

Object Detection

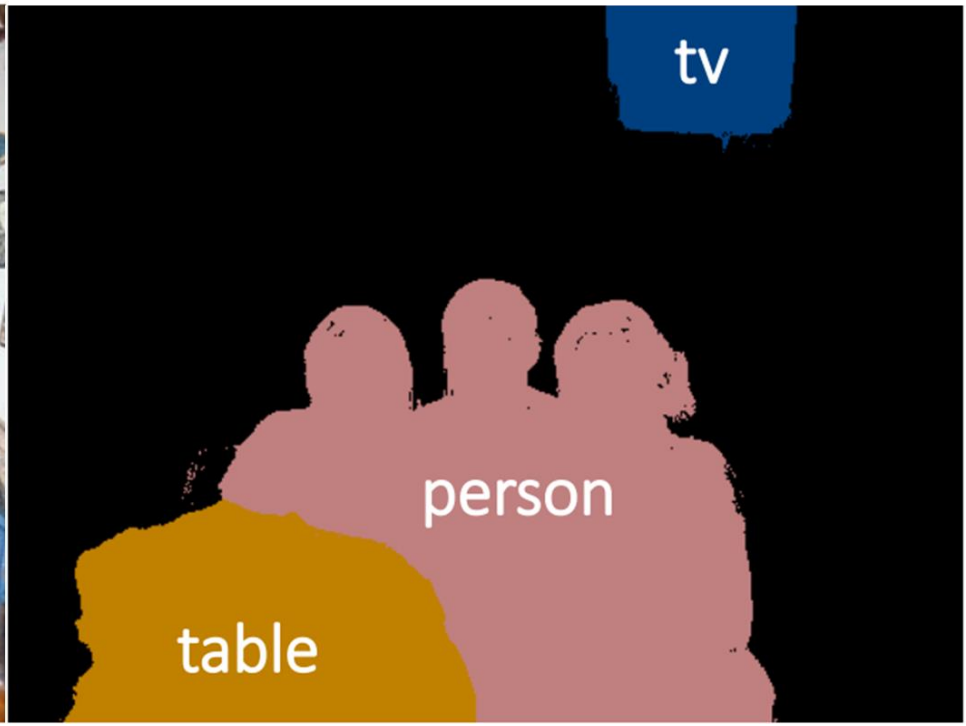




Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, arXiv:1506.01497.

Applications of CNNs

Semantic Image Segmentation



Jifeng Dai, Kaiming He, Jian Sun, BoxSup: Exploiting Bounding Boxes to Supervise Convolutional Networks for Semantic Segmentation, arXiv:1503.01640.

Video Classification, Object Tracking, Optical Flow, Compression Artifacts Reduction, Blur Removal, Deep Edge-Aware Filtering, Deep Stereo Matching, Edge Detection, ……

Assignment Sheet: Introduction

Deep Convolutional Neural Networks for Scene Classification

Convolutional neural networks (CNN) have shown to be valuable for many different tasks like natural language processing as well as image-based object and scene recognition. In order to learn more about CNN's, their properties and possible architectures, you have to train, test and adjust a given CNN. For the task of scene recognition, we provide representative images from 15 different classes: 'bedroom', 'coast', 'forest', 'highway', 'livingroom', 'mountain', 'office', 'opencountry', 'store', 'street', 'suburb' and 'tallbuilding' (Figure 1).

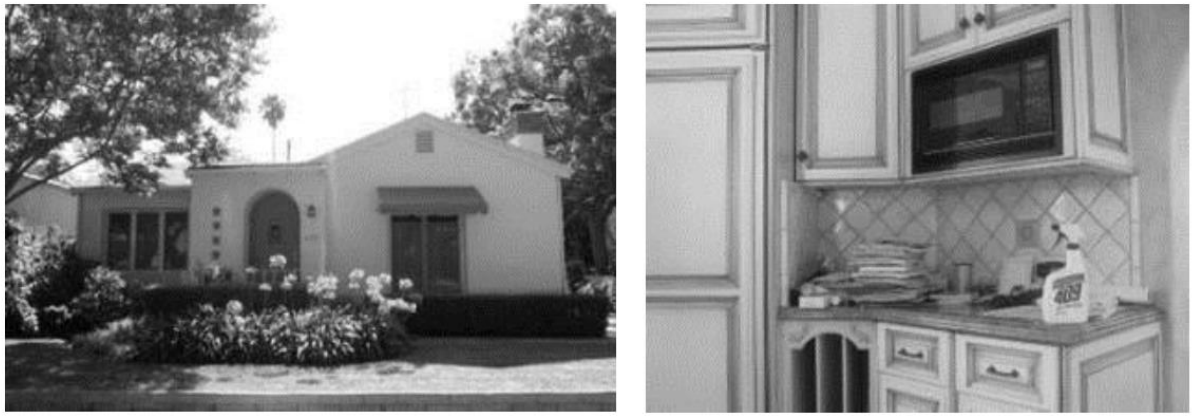


Figure 1: Example images for classes 'suburb' and 'kitchen'

The goal is to maximize the CNN classification accuracy using validation data, i.e., data which was not used for training. Try to improve the CNN so that you reach a classification accuracy around 70 %.

Assignment Sheet: Preparation

1. **Install MatConvNet library** (only available for MATLAB, version 1.0-beta20) from

   http://www.vlfeat.org/matconvnet/

   For installing and compiling simply follow the instructions on

   http://www.vlfeat.org/matconvnet/install/#installing-and-compiling-the-library

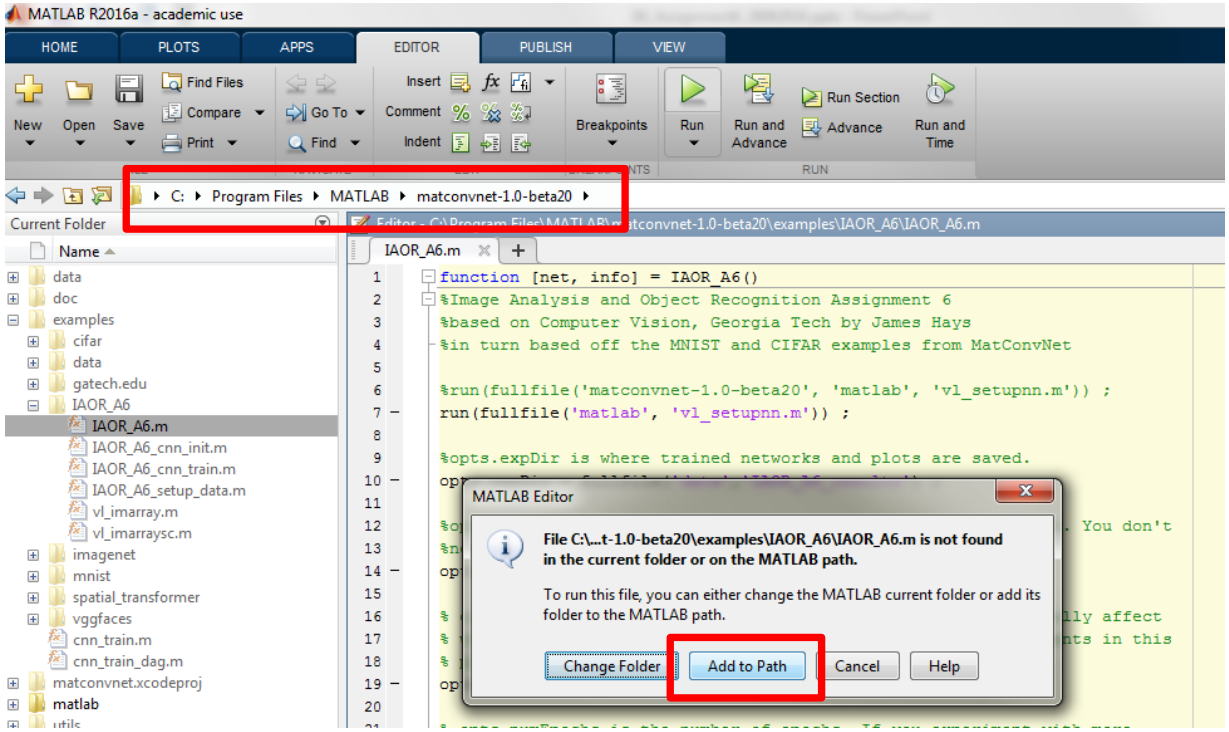
```
> cd <MatConvNet>
> addpath matlab
> savepath
> vl_compilenn
```

2. **Download the starter code** as well as the required dataset from our homepage (password: IAOR.login2017). Copy the folder `15SceneData` to `matconvnet-1.0-beta20\data`. Copy the provided source files to `matconvnet-1.0-beta20\examples\IAOR_A6`. The dataset contains 15 different scene classes. For each class 100 training and even more test images are provided.

3. **Run MATLAB** as administrator (`mkdir` is called) and change to your `matconvnet` folder (`cd`).

Assignment Sheet: Preparation

- The provided starter code…
  - …is a ready-to-run script (`IAOR_A6.m`)
  - …unfortunately can only be used in MATLAB (MatConvNet only available for MATLAB)
  - …performs a training of a pre-defined simple CNN
- Follow instructions (slide before):



Assignment Sheet: CNN Structure

IAOR_A6_cnn_init.m

```
% constant scalar for the random initial network weights. You shouldn't
% need to modify this.
f=1/100;

net.layers = {} ;

net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(9,9,1,10, 'single'), zeros(1, 10, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0, ...
                           'name', 'conv1') ;

net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [7 7], ...
                           'stride', 7, ...
                           'pad', 0) ;

net.layers{end+1} = struct('type', 'relu') ;

net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(8,8,10,15, 'single'), zeros(1, 15, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0, ...
                           'name', 'fc1') ;

% Loss layer
net.layers{end+1} = struct('type', 'softmaxloss') ;

% Visualize the network
vl_simplenn_display(net, 'inputSize', [64 64 1 50])
```

Convolution Layer

Max Pooling

Activation Function: $r(z) = \max(0, z)$

Convolution Layer

Softmaxloss (only for training)

Assignment Sheet: CNN Processing

```
vl_simplenn_display(net, 'inputSize', [64 64 1 50]);
```

layer	0	1	2	3	4	5
type	input	conv	mpool	relu	conv	softmxl
name	n/a	conv1			fc1	
support	n/a	9	7	1	8	1
filt dim	n/a	1	n/a	n/a	10	n/a
num filts	n/a	10	n/a	n/a	15	n/a
stride	n/a	1	7	1	1	1
pad	n/a	0	0	0	0	0
rf size	n/a	9	15	15	64	64
rf offset	n/a	5	8	8	32.5	32.5
rf stride	n/a	1	7	7	7	7
data size	64	56	8	8	1	1
data depth	1	10	10	10	15	1
data num	50	50	50	50	50	1
data mem	800KB	6MB	125KB	125KB	3KB	4B
param mem	n/a	3KB	0B	0B	38KB	0B

parameter memory|41KB (1e+04 parameters)|
data memory|7MB (for batch size 50)|

- softmaxloss required for training

- 64x64 pix input images
- One channel (grayscale)
- 50 images each batch

- 1x1 spatial resolution
- Data depth = number of classes

→ **Vector of 15 values between 0,..,1 which describe result class membership**

Assignment Sheet: Run IAOR_A6.m

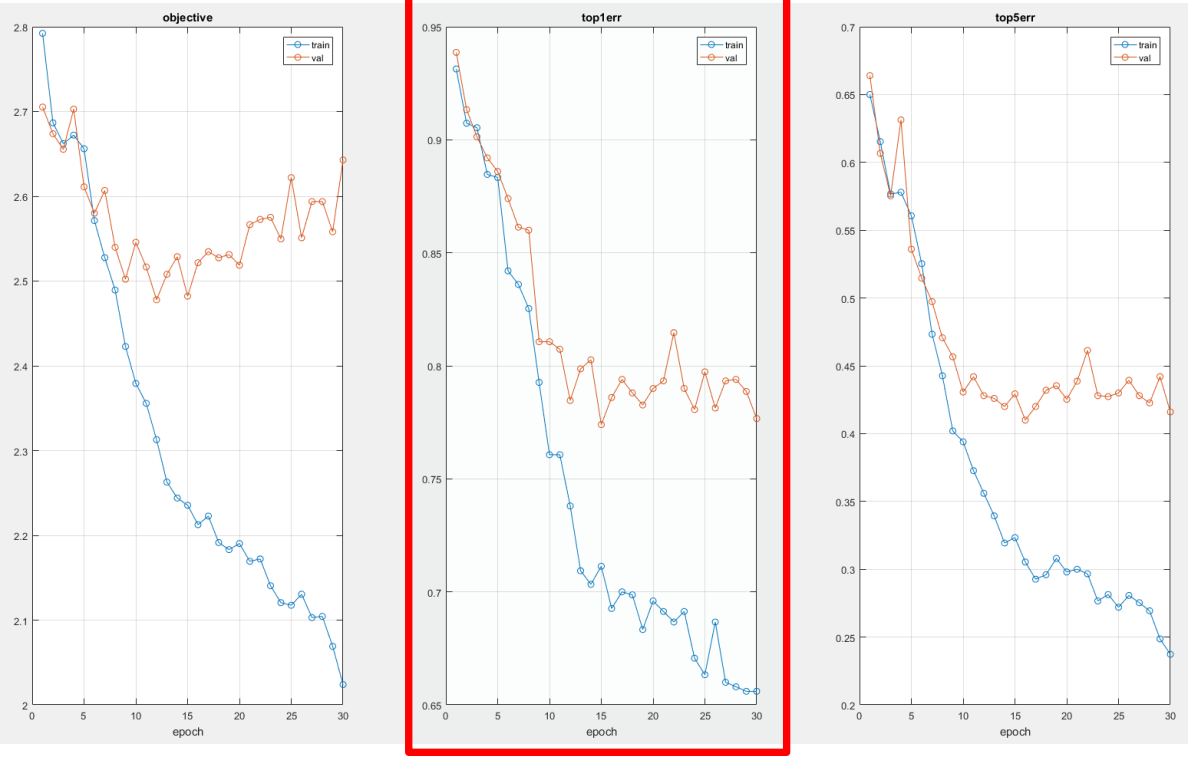
4. **Run the main fuction** (`IAOR_A6.m`) to train (and also test) the provided CNN from scratch. The training will produce three different plots (Figure 2). In each training epoch, the 1500 training images are randomly subdivided into batches of 50 images. A final "loss layer" backpropagates the estimated loss (the true class memberships are known) in each iteration and the weights are updated correspondingly. The top1 error describes, how often the highest scoring guess is wrong.

- **CNN Training:**
  - Images with known class membership are used for training (15SceneData\train)
  - Another set of „unseen" test images (15SceneData\test) is used for validation

Assignment Sheet: Run IAOR_A6.m

4. **Run the main fuction** (`IAOR_A6.m`) to train (and also test) the provided CNN from scratch. The training will produce three different plots (Figure 2). In each training epoch, the 1500 training images are randomly subdivided into batches of 50 images. A final "loss layer" backpropagates the estimated loss (the true class memberships are known) in each iteration and the weights are updated correspondingly. The top1 error describes, how often the highest scoring guess is wrong.

Error measure



Accuracies using train and val datasets

Assignment Sheet: Run IAOR_A6.m

4. **Run the main fuction** (`IAOR_A6.m`) to train (and also test) the provided CNN from scratch. The training will produce three different plots (Figure 2). In each training epoch, the 1500 training images are randomly subdivided into batches of 50 images. A final "loss layer" backpropagates the estimated loss (the true class memberships are known) in each iteration and the weights are updated correspondingly. The top1 error describes, how often the highest scoring guess is wrong. Answer the following questions:
    - **a.** Describe the axes in brief.
    - **b.** What is the reason for the differences of the graphs train and val?
    - **c.** Increase the number of epochs. What can be observed?

Assignment Sheet: Run IAOR_A6.m

Editor - C:\Program Files\MATLAB\matconvnet-1.0-beta20\examples\IAOR_A6\IAOR_A6.m

IAOR_A6.m | IAOR_A6_cnn_init.m

```
1  function [net, info] = IAOR_A6()
2  %Image Analysis and Object Recognition Assignment 6
3  %based on Computer Vision, Georgia Tech by James Hays
4  %in turn based off the MNIST and CIFAR examples from MatConvNet
5
6  %run(fullfile('matconvnet-1.0-beta20', 'matlab', 'vl_setupnn.m')) ;
7  run(fullfile('matlab', 'vl_setupnn.m')) ;
8
9  %opts.expDir is where trained networks and plots are saved.
10 opts.expDir = fullfile('data','IAOR_A6_results') ;
11
12 %opts.batchSize is the number of training images in each batch. You don't
13 %need to modify this.
14 opts.batchSize = 50 ;
15
16 % opts.learningRate is a critical parameter that can dramatically affect
17 % whether training succeeds or fails. For most of the experiments in this
18 % project the default learning rate is safe.
19 opts.learningRate = 0.0001 ;
20
21 % opts.numEpochs is the number of epochs. If you experiment with more
22 % complex networks you might need to increase this. Likewise if you add
23 % regularization that slows training.
24 opts.numEpochs = 30 ;
```

Assignment Sheet: Run IAOR_A6.m

4. **Run the main fuction** (`IAOR_A6.m`) to train (and also test) the provided CNN from scratch. The training will produce three different plots (Figure 2). In each training epoch, the 1500 training images are randomly subdivided into batches of 50 images. A final "loss layer" backpropagates the estimated loss (the true class memberships are known) in each iteration and the weights are updated correspondingly. The top1 error describes, how often the highest scoring guess is wrong. Answer the following questions:

    d. After the training in `IAOR_A6()` we are interested, if the CNN really performs as bad as shown in the plots. Test the trained CNN by hand using some randomly chosen images of the validation set. Hints:

    i. The training was done using down sampled images (64x64 pix), so your test images need the same pre-pocessing (`imresize`).

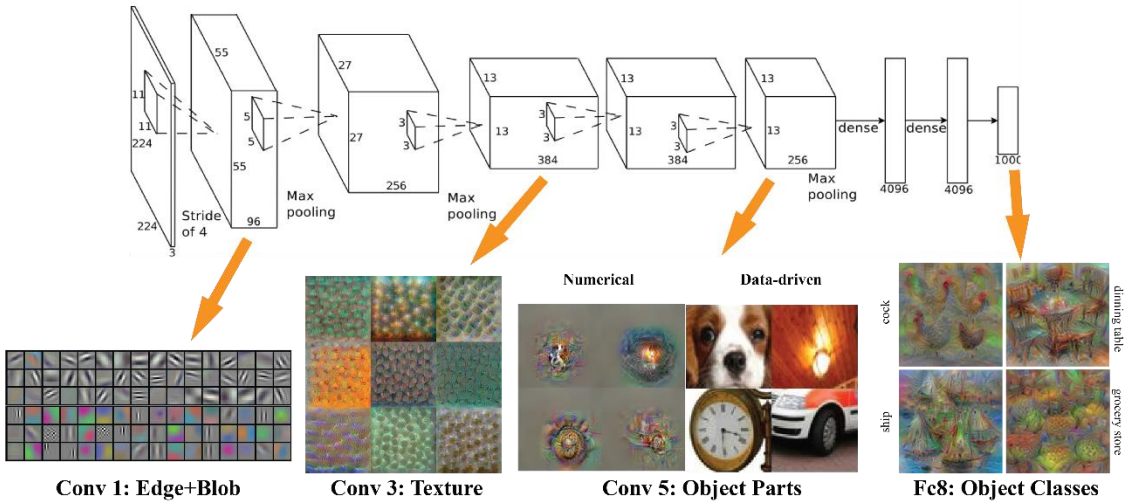
    ii. The final layer type is "`softmaxloss`", which is only needed for training. The type has to be changed before testing, for example with

    ```
    net.layers{end}.type = 'softmax';
    ```

    iii. Use the function `vl_simplenn` for testing. The 15-element score vector of the final layer can be obtained by `scores = squeeze(gather(res(end).x))`

Assignment Sheet: Understanding

5. **Understanding the CNN**: The following layer types are used here: *convolution, max pooling* and *non-linearity* (RELU: rectified linear unit). Furthermore, after the initialization of the net in `IAOR_A6_cnn_init()`, the architecture is listed in the MATLAB Command Window using the function `vl_simplenn_display`. Use this information to describe how an image is processed by the CNN. What is the input and output of each layer? You may also try to visualize the process, like done here (http://vision03.csail.mit.edu/cnn_art/index.html).



Assignment Sheet: Improvement

6. **Improving the classification accuracy**: Unfortunately, the results are really as bad as shown in Figure 2. We now try to improve the performance step by step. Choose **at least three** of the five strategies proposed below. Document all steps and the corresponding achieved improvements!

   a. **Number of training images**: We don't have more training images available. A common strategy to synthetically increase the number of images is to transform them. Test the impact of using randomly flipped training image sets in `getBatch()` using the functions `rand` and `fliplr`. Because the training and test errors decrease more slowly now, you may need more training epochs and/or you may try modifying the learning rate.

   b. **Zero-centering of images**: Simply subtract the mean gray-value from each image in order to obtain a zero-centered image. This is a very useful step in pre-processing. Explain why!

Assignment Sheet: Improvement

6. **Improving the classification accuracy**: Unfortunately, the results are really as bad as shown in Figure 2. We now try to improve the performance step by step. Choose **at least three** of the five strategies proposed below. Document all steps and the corresponding achieved improvements!

   c. **Regularization of training**: Especially with a high number of epochs, the train error tends to converge to zero while the val top1 error hovers around 40-50 %. This is caused by overfitting, i.e., our CNN works quite good for the training data but cannot be used for unseen data of the same class (bad generalization). Since the number of available training data is limited here, so-called dropout regularization can be used instead. Add the following additional layer directly before the last convolutional layer:

```
net.layers{end+1} = struct('type', 'dropout', 'rate', 0.5);
```

   This randomly turns off network connections at training time to reduce overfitting and therefore prevents a unit in one layer from relying too strongly on a single unit in the previous layer. The dropout rate describes the proportion of connections that are randomly deleted.

Assignment Sheet: Improvement

6. **Improving the classification accuracy**: Unfortunately, the results are really as bad as shown in Figure 2. We now try to improve the performance step by step. Choose **at least three** of the five strategies proposed below. Document all steps and the corresponding achieved improvements!

   d. **Increasing the deepness of the initial CNN**: Only two layers of our net actually contain learned weights. Other proposed nets contain up to 19 of these layers. However, arranging and parameterize them is not an easy task. Add more layers to the net. A common approach is to use a combination of a "convolutional", "max-pool" and "relu" layer, like already done in the starter code. Caution: The weights, pool and stride variables have to be adjusted, so that the data size in final layer (the one before the softmax layer) is 1 and data depth is 15 (see table output of `vl_simplenn_display`). In general you have to make sure that the data depth output by any channel matches the data depth input to the following channel. Hint: the max-pooling operation covers a window of 7x7 and then is subsampled with a stride of 7. Networks usually do not subsample by more than a factor of 2 or 3 each layer. This might be one parameter to adjust.

Assignment Sheet: Improvement

6. **Improving the classification accuracy**: Unfortunately, the results are really as bad as shown in Figure 2. We now try to improve the performance step by step. Choose **at least three** of the five strategies proposed below. Document all steps and the corresponding achieved improvements!

    e. **Image size**: All images are resized to 64x64 pixels for training and testing which is good in terms of computational time but actually reduces the level of details drastically. Adjust the whole network in order to process full-size or half-size images.