Audrija Mukherjee

Project 6: Deep Learning

The aim here is to design and train deep convolutional networks for scene recognition using the MatConvNet toolbox.In project 4, we used bag of features representation on 15-way scene classification and the aim here is to ultimately outperform the the accuracy obtained then. There are 2 broad parts of this project:

1. Part 1: Training a deep network from scratch
2. Part 2: Finetuning a pretrained deep network

Algorithm Description

Part 1: Training a deep network from scratch

The starter code has the following layers-

A convolutional layer- The weights initialized with random numbers from a Gaussian distribution. The filters have a 9x9 spatial resolution, span 1 filter depth and there are 10 filters.

A max pool layer-It will take a max over a 7x7 sliding window and then subsample the resulting image / map with a stride of 7. Thus the max pooling layer will decrease the spatial resolution by a factor of 7 according to the stride parameter. The filter depth will remain the same (10).

A rectified linear layer- Any values in the feature map from the max pooling layer which are negative will be set to 0. This ;ayer brings the non-linearity to the network

A fully connected convolutional layer- The filters learned at this layer operate on the rectified, subsampled, maxpooled filter responses from the first layer. The output of this layer must be 1x1 spatial resolution (or "data size") and it must have a filter depth of 15 (corresponding to the 15 categories of the 15 scene database). 8x8 is the spatial resolution of the filters and 10 is the number of filter dimensions that each of these filters take as input and 15 is the number of dimensions out.

At the top of our network we add one more layer which is only used for training. This is the softmax "loss" layer. This loss function will measure how badly the network is doing for any input. The network weights will update, through backpropagation, based on the derivative of the loss function.

Details for starter code network architecture

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)

Problem 1: Not enough training data. Solution: Jitter

When we leftright flip (mirror) an image of a scene, it never changes categories. A kitchen doesn't become a forest when mirrored. So, to increase the number of training images, I created a 4-D matrix of batch size number of original images and their mirror images. Now, the total number of images in the 4-D matrix is twice the number of batch size. Since we want to return batch size number of images from getBatch(), I randomly chose batch size number of these images i.e. some of the original images are returned and rest are flipped versions of the remaining original images.

Problem 2: The images aren't zero-centered. Solution: Zero-center them

In proj6_part1_setup_data.m, I computed the mean image from all images in imdb.data and then subtracted it from all images before returning imdb. Thus, the images are now zero-centred.

Problem 3: Our network isn't regularized. Solution: Add a dropout layer

After fixing, problem 1 and 2, we see that the network is overfitting to the training data. To avoid that, we add a dropout layer in proj6_part1_cnn_init() directly before the last convolutional layer. Dropout regularization randomly turns off network connections at training time to fight overfitting. This prevents a unit in one layer from relying too strongly on a single unit in the previous layer. Dropout regularization can be interpreted as simultaneously training many "thinned"versions of the network. At test time, all connections are restored which is analogous to taking an average prediction over all of the "thinned" networks.

Problem 4: Our network isn't deep. Solution: Add a few layers

In proj6_part1_cnn_init(), after the existing relu layer, I added another convolutional layer, maxpool layer and rectified linear(relu) layer. We need the network's final layer (not counting the softmax) to have a data size of 1 and a data depth of 15. Also, for the max-pooling layer, we do not want a stride of 7, which the original baseline specifies. Therefore, we reduce it. I specified the layers of my network as follows:

Details for Part 1 network architecture

```
     layer|    0|    1|    2|    3|    4|    5|    6|      7|    8|      9|
      type|input| conv|mpool| relu| conv|mpool| relu|dropout| conv|softmxl|
      name|  n/a|conv1|     |     |conv2|     |     |       |conv3|       |
----------|-----|-----|-----|-----|-----|-----|-----|-------|-----|-------|
   support|  n/a|    9|    7|    1|    5|    3|    1|      1|    6|      1|
  filt dim|  n/a|    1|  n/a|  n/a|   10|  n/a|  n/a|    n/a|   15|    n/a|
 num filts|  n/a|   10|  n/a|  n/a|   15|  n/a|  n/a|    n/a|   15|    n/a|
    stride|  n/a|    1|    3|    1|    1|    2|    1|      1|    1|      1|
       pad|  n/a|    0|    0|    0|    0|    0|    0|      0|    0|      0|
----------|-----|-----|-----|-----|-----|-----|-----|-------|-----|-------|
   rf size|  n/a|    9|   15|   15|   27|   33|   33|     33|   63|     63|
 rf offset|  n/a|    5|    8|    8|   14|   17|   17|     17|   32|     32|
 rf stride|  n/a|    1|    3|    3|    3|    6|    6|      6|    6|      6|
----------|-----|-----|-----|-----|-----|-----|-----|-------|-----|-------|
 data size|   64|   56|   17|   17|   13|    6|    6|      6|    1|      1|
data depth|    1|   10|   10|   10|   15|   15|   15|     15|   15|      1|
  data num|   50|   50|   50|   50|   50|   50|   50|     50|   50|      1|
----------|-----|-----|-----|-----|-----|-----|-----|-------|-----|-------|
  data mem|800KB|  6MB|564KB|564KB|495KB|105KB|105KB|  105KB|  3KB|     4B|
 param mem|  n/a|  3KB|   0B|   0B| 15KB|   0B|   0B|     0B| 32KB|     0B|

parameter memory|50KB (1.3e+04 parameters)|
     data memory|  9MB (for batch size 50)|
```

1. Convolutional layer: 9X9 spatial resolution, 10 filter depth
2. Maxpool layer: 7x7 sliding window, stride 3
3. Rectified Linear layer
4. Convolutional layer: 5X5 spatial resolution, 15 filter depth
5. Maxpool layer: 3x3 sliding window, stride 2
6. Rectified Linear layer
7. Dropout layer: Dropout rate 0.5
8. Convolutional layer: 6X6 spatial resolution, 15 filters
9. Softmax loss layer

NOTE: Results for each step are explained in the results section below.

Part 2: Finetuning a pretrained deep network

The representations learned by deep convolutional networks generalize surprisingly well to other recognition tasks. We use this to finetune the VGGF network to perform scene recognition. The original VGGF network had the following architecture.

Details for original VGGF network architecture

```
     layer|    0|    1|    2|    3|     4|    5|    6|    7|    8|    9|   10|   11|   12|   13|   14|   15|   16|   17|   18|   19|   20|    21|
      type|input| conv| relu| norm|  mpool| conv| relu| norm|mpool| conv| relu| conv| relu| conv| relu|mpool| conv| relu| conv| relu| conv|softmx|
      name|  n/a|conv1|relu1|norm1|  pool1|conv2|relu2|norm2|pool2|conv3|relu3|conv4|relu4|conv5|relu5|pool5|  fc6|relu6|  fc7|relu7|  fc8|  prob|
----------|-----|-----|-----|-----|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
   support|  n/a|   11|    1|    1|      3|    5|    1|    1|    3|    3|    1|    3|    1|    3|    1|    3|    6|    1|    1|    1|    1|     1|
  filt dim|  n/a|    3|  n/a|  n/a|    n/a|   64|  n/a|  n/a|  n/a|  256|  n/a|  256|  n/a|  256|  n/a|  n/a|  256|  n/a| 4096|  n/a| 4096|   n/a|
 num filts|  n/a|   64|  n/a|  n/a|    n/a|  256|  n/a|  n/a|  n/a|  256|  n/a|  256|  n/a|  256|  n/a|  n/a| 4096|  n/a| 4096|  n/a| 1000|   n/a|
    stride|  n/a|    4|    1|    1|      2|    1|    1|    1|    2|    1|    1|    1|    1|    1|    1|    2|    1|    1|    1|    1|    1|     1|
       pad|  n/a|    0|    0|    0|0x1x0x1|    2|    0|    0|    0|    1|    0|    1|    0|    1|    0|    0|    0|    0|    0|    0|    0|     0|
----------|-----|-----|-----|-----|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
   rf size|  n/a|   11|   11|   11|     19|   51|   51|   51|   67|   99|   99|  131|  131|  163|  163|  195|  355|  355|  355|  355|  355|   355|
 rf offset|  n/a|    6|    6|    6|     10|   10|   10|   10|   18|   18|   18|   18|   18|   18|   18|   34|  114|  114|  114|  114|  114|   114|
 rf stride|  n/a|    4|    4|    4|      8|    8|    8|    8|   16|   16|   16|   16|   16|   16|   16|   32|   32|   32|   32|   32|   32|    32|
----------|-----|-----|-----|-----|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
 data size|  224|   54|   54|   54|     27|   27|   27|   27|   13|   13|   13|   13|   13|   13|   13|    6|    1|    1|    1|    1|    1|     1|
data depth|    3|   64|   64|   64|     64|  256|  256|  256|  256|  256|  256|  256|  256|  256|  256|  256| 4096| 4096| 4096| 4096| 1000|  1000|
  data num|   50|   50|   50|   50|     50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|   50|    50|
----------|-----|-----|-----|-----|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
  data mem| 29MB| 36MB| 36MB| 36MB|    9MB| 36MB| 36MB| 36MB|  8MB|  8MB|  8MB|  8MB|  8MB|  8MB|  8MB|  2MB|800KB|800KB|800KB|800KB|195KB| 195KB|
 param mem|  n/a| 91KB|   0B|   0B|     0B|  2MB|   0B|   0B|   0B|  2MB|   0B|  2MB|   0B|  2MB|   0B|   0B|144MB|   0B| 64MB|   0B| 16MB|    0B|

parameter memory|232MB (6.1e+07 parameters)|
     data memory| 314MB (for batch size 50)|
```

I made the following edits to the existing network in proj6_part2_cnn_init.m:

1. Removed fc8 and the softmax layer
2. Redefined fc8 to have 1X1 spatial resolution, input filter depth 4096 and output filter depth 15
3. Added dropout layers between fc6 and fc7 and between fc7 and fc8
4. Redefined Softmax loss layer, the top layer of the network

Details of modified VGGF network architecture

layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
type	input	conv	relu	norm	mpool	conv	relu	norm	mpool	conv	relu	conv	relu	conv	relu	mpool	conv	relu	dropout	conv	relu	dropout	conv	so
name	n/a	conv1	relu1	norm1	pool1	conv2	relu2	norm2	pool2	conv3	relu3	conv4	relu4	conv5	relu5	pool5	fc6	relu6		fc7	relu7		fc8
support	n/a	11	1	1	3	5	1	1	3	3	1	3	1	3	1	3	6	1	1	1	1	1	1
filt dim	n/a	3	n/a	n/a	n/a	64	n/a	n/a	n/a	256	n/a	256	n/a	256	n/a	n/a	256	n/a	n/a	4096	n/a	n/a	4096
num filts	n/a	64	n/a	n/a	n/a	256	n/a	n/a	n/a	256	n/a	256	n/a	256	n/a	n/a	4096	n/a	n/a	4096	n/a	n/a	15
stride	n/a	4	1	1	2	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1
pad	n/a	0	0	0	0x1x0x1	2	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
rf size	n/a	11	11	11	19	51	51	51	67	99	99	131	131	163	163	195	355	355	355	355	355	355	355
rf offset	n/a	6	6	6	10	10	10	10	18	18	18	18	18	18	18	34	114	114	114	114	114	114	114
rf stride	n/a	4	4	4	8	8	8	8	16	16	16	16	16	16	16	32	32	32	32	32	32	32	32
data size	224	54	54	54	27	27	27	27	13	13	13	13	13	13	13	6	1	1	1	1	1	1	1
data depth	3	64	64	64	64	256	256	256	256	256	256	256	256	256	256	256	4096	4096	4096	4096	4096	4096	15
data num	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
data mem	29MB	36MB	36MB	36MB	9MB	36MB	36MB	36MB	8MB	8MB	8MB	8MB	8MB	8MB	8MB	2MB	800KB	800KB	800KB	800KB	800KB	800KB	3KB
param mem	n/a	91KB	0B	0B	0B	2MB	0B	0B	0B	2MB	0B	2MB	0B	2MB	0B	0B	144MB	0B	0B	64MB	0B	0B	240KB

```
parameter memory|217MB (5.7e+07 parameters)|
     data memory| 315MB (for batch size 50)|
```

proj6_part2_setup_data.m is the same as part 1, except a few changes. VGGF reuires input images to be of size 224x224. So I resized the input images to 224x224. VGGF accepts 3 channel images. Since our inputs are grayscale, I replicated the grayscale values over all three channels by concatenating the grayscale images with themselves to make an RGB(3 channel) image.

Example of code with highlighting

```
%example code- Part 1 jittering-getBatch()
flip_im=fliplr(im);
im=cat(4,im,flip_im);
rand_indices=randi(2*size_batch,size_batch,1);
```

```
%example code- Part 1 zero centering- proj6_part1_setup_data.m
mean_img=mean(all_images,4);
for i =1:size(all_images,4)
    all_images(:,:,:,i)=all_images(:,:,:,i)-mean_img;
end
```

```
%example code- Part 1 Regularizing- proj6_part1_cnn_init.m
net.layers{end+1} = struct('type', 'dropout','rate',0.5);
```

```
%example code- Part 1 Deeper Network- proj6_part1_cnn_init.m
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,10,15, 'single'), zeros(1, 15, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0, ...
                           'name', 'conv2') ;

net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [3 3], ...
                           'stride', 2, ...
                           'pad', 0) ;

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

```
%example code- Part 2: VGGF modification- proj6_part1_cnn_init.m
net.layers{1,21} = struct('type', 'dropout','rate',0.5);
net.layers{1,22} = struct('type', 'conv', ...
                          'weights', {{f*randn(1,1,4096,15, 'single'), zeros(1, 15, 'single')}}, ...
                          'stride', 1, ...
                          'pad', 0, ...
                          'name', 'fc8') ;
```

RESULTS

Number of Paramters

Part 1- Change in number of parameters

As we modify the network architecture, the total number of parameters change. In part 1- Problem 4, we added a convolutional, max-pool and rectified linear layer and the total number of parameters for a batch size of 50 increased. The table below shows the change in number of parameters

Network	No. of parameters
Original Baseline	1e+04
Deeper Network- Part 1, Problem 4	1.3e+04

Part 2- Change in number of parameters

As we modify the VGGF network architecture to finetune it for scene classification, the total number of parameters change. In part 2, I modified a convolutional layer and loss layer, and added 2 dropout layers. The output filter depth for the last convolutional layer is much lower for the modified version of network architecture and the total number of parameters for a batch size of 50 decreased.The table below shows the change in number of parameters

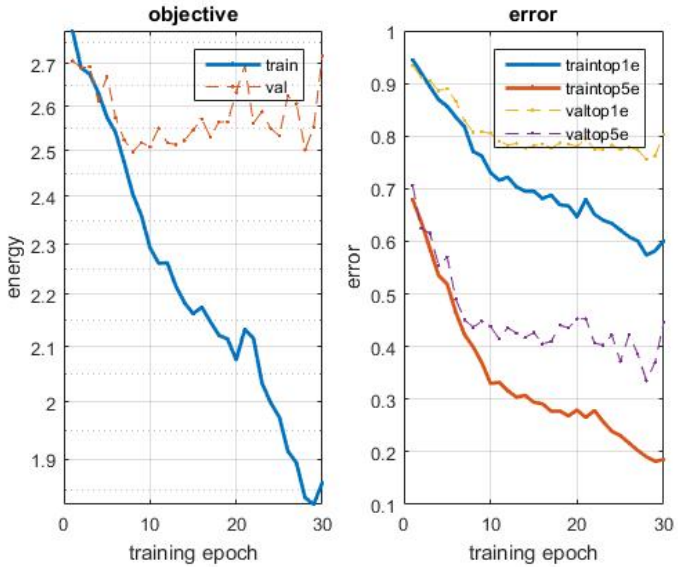
Network	No. of parameters
VGGF	6.1e+07
Modified VGGF- Part 2	5.7e+07

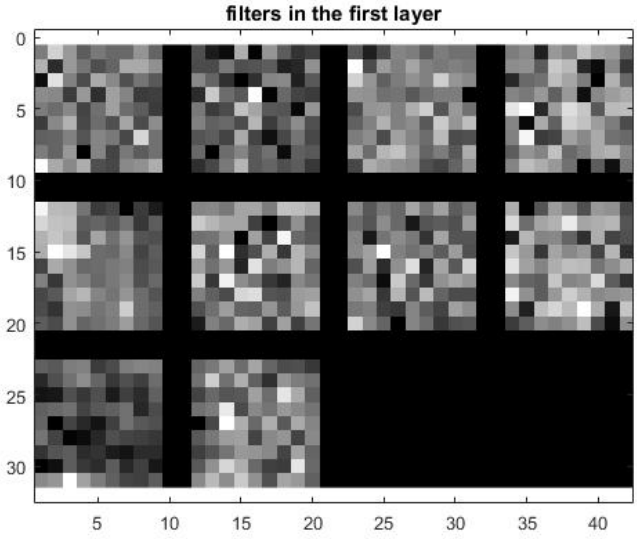
Lowest validation errors and 1st layer filters

Summary of lowest validation errors

Part	Epochs	Lowest validation Error	Accuracy
Original Baseline	30	0.756000	0.244000
After Part 1- Problem 1: Jittering	30	0.685333	0.314667
	50	0.652000	0.348000
After Part 1- Problem 2: Zero centering	50	0.480000	0.520000
After Part 1- Problem 3: Regularizing	50	0.473333	0.526667
	80	0.430000	0.570000
After Part 1- Problem 4: Adding Layers	120	0.434667	0.565333
After Part 2: Modifying VGGF	10	0.122000	0.878000

Original Baseline Results

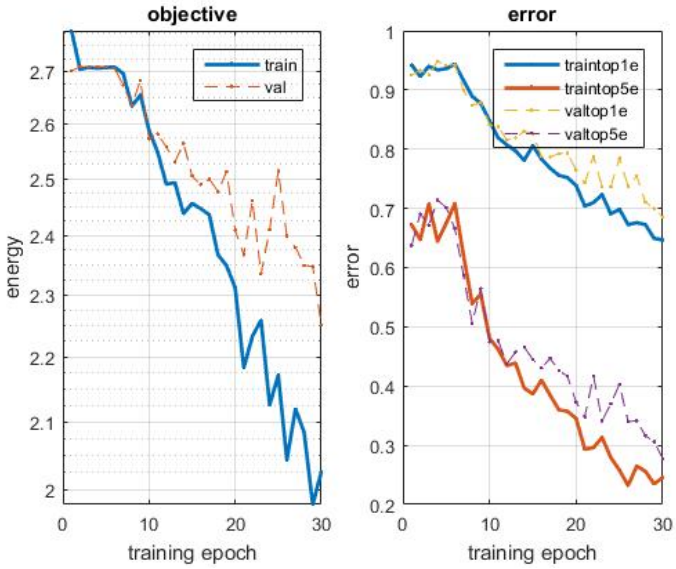




Lowest validation error is 0.756000

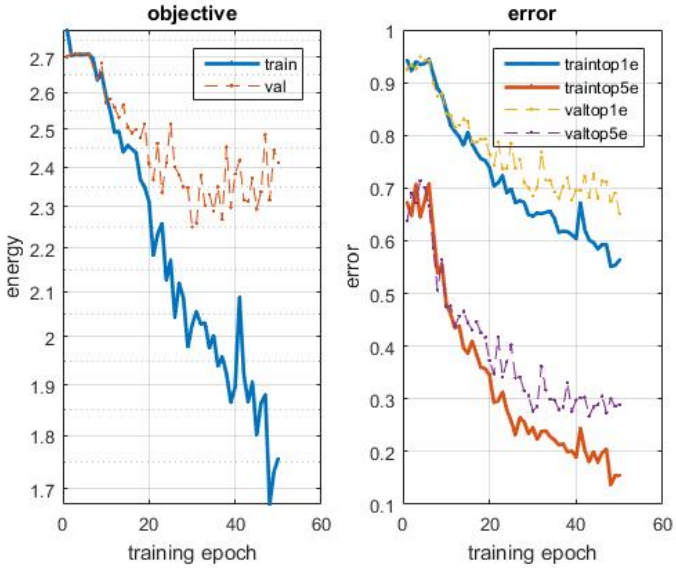
Part 1 Results

Results after solving Problem 1- 30 Epochs



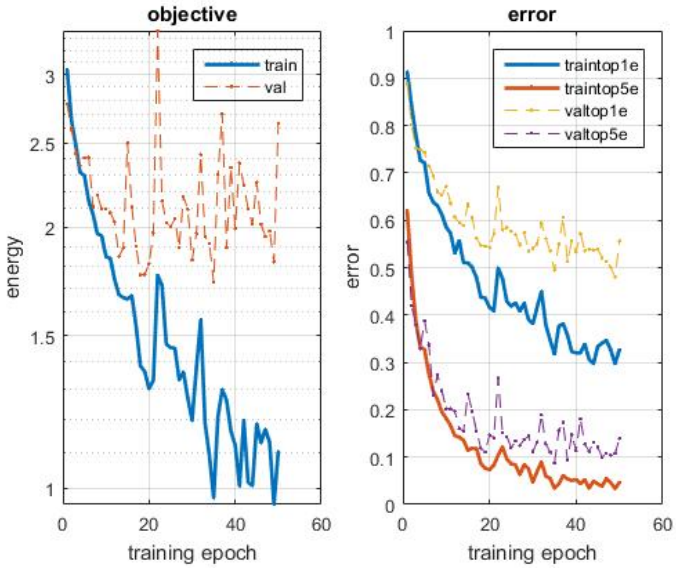
Lowest validation error is 0.685333

Results after solving Problem 1- 50 Epochs



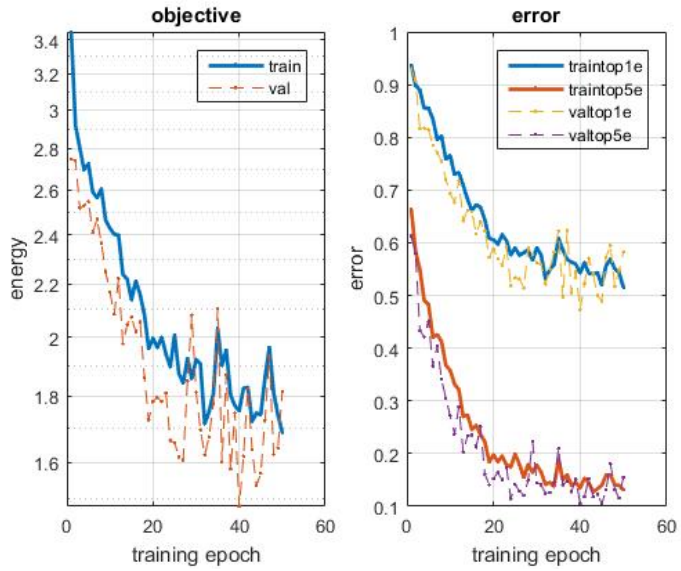
Lowest validation error is 0.652000

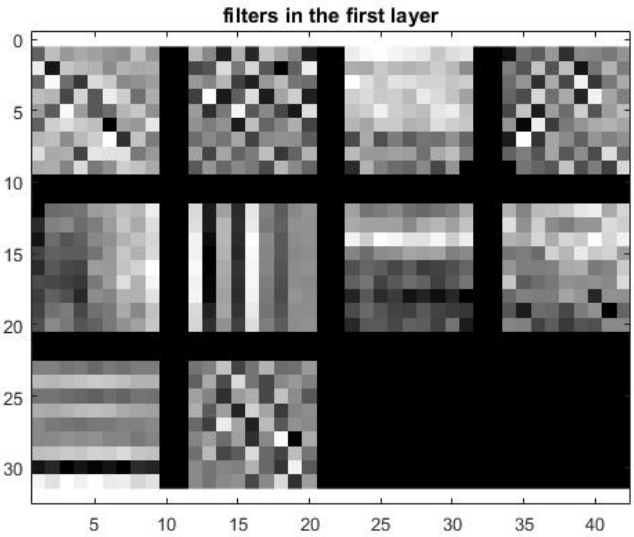
Results after solving Problem 2- 50 Epochs



Lowest validation error is 0.480000

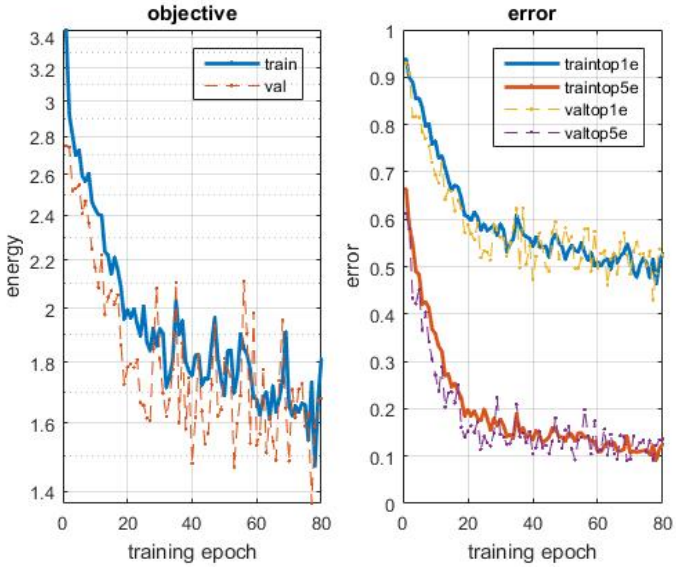
Results after solving Problem 3- 50 Epochs





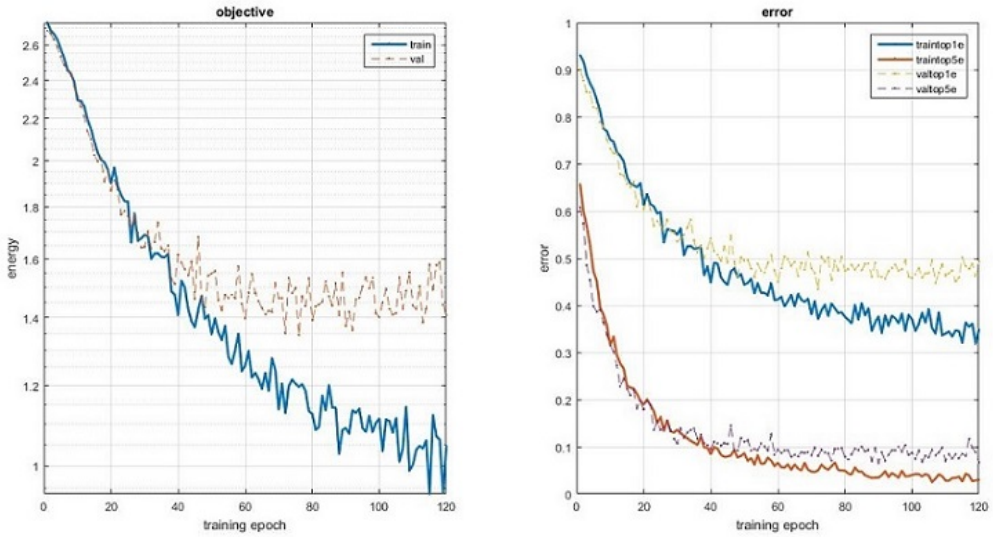
Lowest validation error is 0.473333

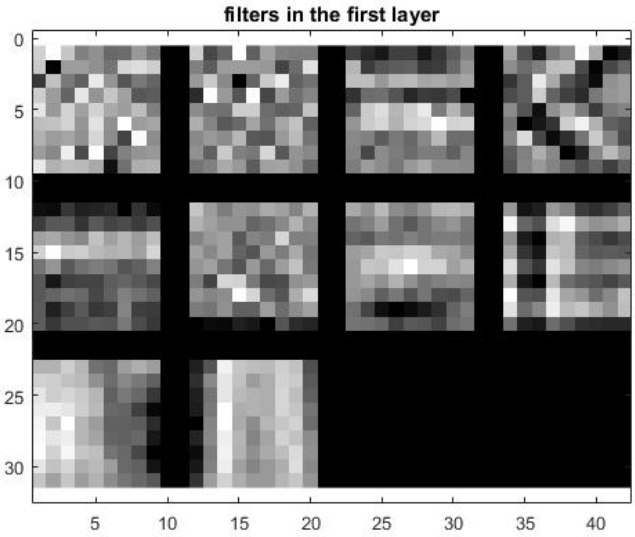
Results after solving Problem 3- 80 Epochs



Lowest validation error is 0.430000. (Remains same after 120 epochs)

Results after solving Problem 4- 120 Epochs

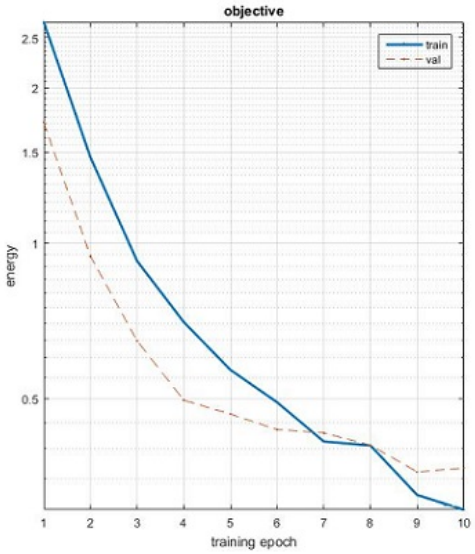


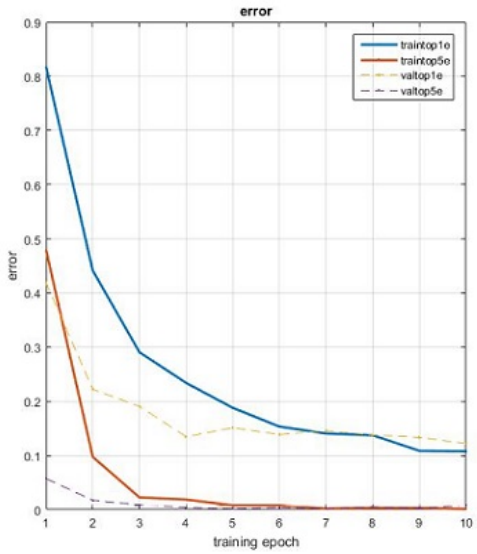


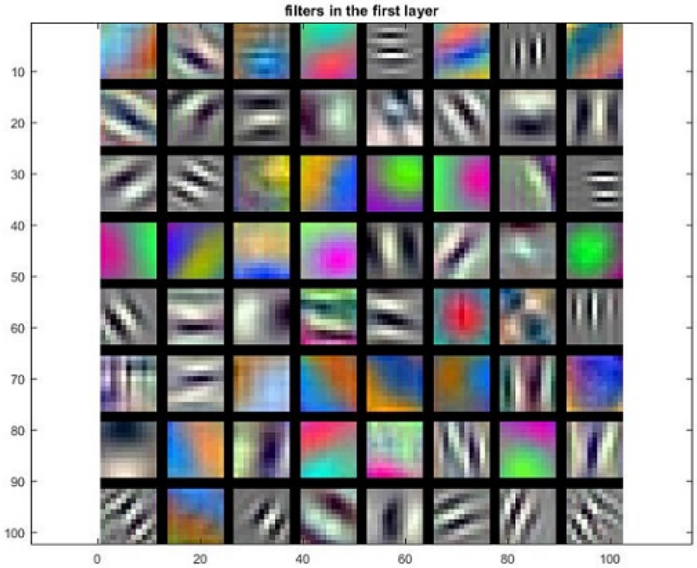
Lowest validation error is 0.434667

Part 2 Results

Results after modifying VGGF network- 10 epochs







Lowest validation error is 0.122