

Preparing Data for Training

We prepare the downloaded images for training. For that we do not need to bring all the data in at once as it would occupy a lot of memory. We can do that efficiently by using datastore and applying augmentation to make it suitable for training.

ImageDataStore

We will create a reference to the folder which contains all the labels including images. That way we would not bring all the images at once in MATLAB. This script should be saved in the folder containing FlowerImages folder where we downloaded the data initially

```
Flowerds = imageDatastore("Pets/", "IncludeSubfolders", true, "LabelSource", "foldernames")
```

```
Flowerds =  
  ImageDatastore with properties:  
    Files: {  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00001.jpg';  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00002.jpg';  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00003.jpg'  
        ... and 7587 more  
    }  
   Folders: {  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets'  
    }  
   Labels: [Bengal; Bengal; Bengal ... and 7587 more categorical]  
AlternateFileSystemRoots: {}  
      ReadSize: 1  
SupportedOutputFormats: ["png"    "jpg"    "jpeg"    "tif"    "tiff"]  
DefaultOutputFormat: "png"  
      ReadFcn: @readDatastoreImage
```

Training and Validation set

We have a total of 8183 images with each label containing at least 40 images. We will divide the data into training and validation set with a proportion of 90 to 10. We will have atleast 4 images in each label for testing which we can increase by creating an augmented data set.

```
[imtrain, imval] = splitEachLabel(Flowerds, 0.9)
```

```
imtrain =  
  ImageDatastore with properties:  
    Files: {  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00001.jpg';  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00002.jpg';  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00003.jpg'  
        ... and 6828 more  
    }  
   Folders: {  
        'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets'  
    }  
   Labels: [Bengal; Bengal; Bengal ... and 6828 more categorical]  
AlternateFileSystemRoots: {}  
      ReadSize: 1  
SupportedOutputFormats: ["png"    "jpg"    "jpeg"    "tif"    "tiff"]
```

```

        DefaultOutputFormat: "png"
        ReadFcn: @readDatastoreImage
imval =
    ImageDatastore with properties:

        Files: {
            'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00181.jpg';
            'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00182.jpg';
            'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets\Bengal\image_00183.jpg'
            ... and 756 more
        }
        Folders: {
            'C:\Users\Admin\MATLAB Drive\Homework_week6\Pets'
        }
        Labels: [Bengal; Bengal; Bengal ... and 756 more categorical]
AlternateFileSystemRoots: {}
ReadSize: 1
SupportedOutputFormats: ["png"    "jpg"    "jpeg"    "tif"    "tiff"]
        DefaultOutputFormat: "png"
        ReadFcn: @readDatastoreImage

```

Augment Training Images

As a rule of thumb, we do not train the network on exact same images in every epoch therefore we ease the process of modifying the images for the entire training by using augmentation. For this purpose we create augmented image data store which takes in the datastore reference to training set. This function will perform the augmentation (modification) to images seamlessly when the network is being trained.

Even if the modification to images is not desired, augmentedImageDatastore can be used to rectify the size of the images on the go as per network demands

```
auds = augmentedImageDatastore([224 224],imtrain,'ColorPreprocessing','gray2rgb')
```

```

auds =
    augmentedImageDatastore with properties:

        NumObservations: 6831
        Files: {6831x1 cell}
AlternateFileSystemRoots: {}
        MiniBatchSize: 128
        DataAugmentation: 'none'
        ColorPreprocessing: 'gray2rgb'
        OutputSize: [224 224]
        OutputSizeMode: 'resize'
DispatchInBackground: 0

```

Total Number of Samples for each label

Number of samples for each label

```

[num,~,ic] = unique(imtrain.Labels);

a_table = table(num,accumarray(ic,1))

```

```
a_table = 38x2 table
```

	num	Var2
1	Bengal	180
2	Birman	180
3	Bombay	180
4	British_Shorthair	180
5	Egyptian Mau	180
6	Maine_Coon	180
7	Persian	180
8	Ragdoll	180
9	Russian Blue	180
10	Siamese	180
11	Sphynx	180
12	abyssinian	180
13	american bulldog	180
14	american pit bul...	180
15	american_pit_bul...	180
16	basset_hound	180
17	beagle	180
18	boxer	180
19	chihuahua	180
20	english cocker s...	180
21	english setter	180
22	german shorthaired	180
23	great pyrenees	180
24	havanese	180
25	japanese chin	180
26	keeshond	180
27	leonberger	180
28	miniature pinsche	180
29	newfoundland	180
30	pomeranian	180
31	pug	180
32	saint bernard	180
33	samoyed	180

	num	Var2
34	scottish terrier	179
35	shiba inu	180
36	staffordshire bu...	172
37	wheaten terrier	180
38	yorkshire terrier	180

```
disp(['Minimum number of training images in a label: ' num2str(min(a_table.Var2))])
```

Minimum number of training images in a label: 172

Augmenting Validation Set

The purpose of augmenting validation set of images to increase the number of images for validation so that we have more than 10 images for each label. In this case we load all the images in the MATLAB memory. We will use this data for validation during training.

```
valset = [readall(augmentedImageDatastore([224 224],imval))
readall(augmentedImageDatastore([224 224],imval,'ColorPreprocessing','gray2rgb'))]
```

valset = 1518x2 table

	input	response
1	224×224×3 uint8	Bengal
2	224×224×3 uint8	Bengal
3	224×224×3 uint8	Bengal
4	224×224×3 uint8	Bengal
5	224×224×3 uint8	Bengal
6	224×224×3 uint8	Bengal
7	224×224×3 uint8	Bengal
8	224×224×3 uint8	Bengal
9	224×224×3 uint8	Bengal
10	224×224×3 uint8	Bengal
11	224×224×3 uint8	Bengal
12	224×224×3 uint8	Bengal
13	224×224×3 uint8	Bengal
14	224×224×3 uint8	Bengal
15	224×224×3 uint8	Bengal
16	224×224×3 uint8	Bengal
17	224×224×3 uint8	Bengal

	input	response
18	224×224×3 uint8	Bengal
19	224×224×3 uint8	Bengal
20	224×224×3 uint8	Bengal
21	224×224×3 uint8	Birman
22	224×224×3 uint8	Birman
23	224×224×3 uint8	Birman
24	224×224×3 uint8	Birman
25	224×224×3 uint8	Birman
26	224×224×3 uint8	Birman
27	224×224×3 uint8	Birman
28	224×224×3 uint8	Birman
29	224×224×3 uint8	Birman
30	224×224×3 uint8	Birman
31	224×224×3 uint8	Birman
32	224×224×3 uint8	Birman
33	224×224×3 uint8	Birman
34	224×224×3 uint8	Birman
35	224×224×3 uint8	Birman
36	224×224×3 uint8	Birman
37	224×224×3 uint8	Birman
38	224×224×3 uint8	Birman
39	224×224×3 uint8	Birman
40	224×224×3 uint8	Birman
41	224×224×3 uint8	Bombay
42	224×224×3 uint8	Bombay
43	224×224×3 uint8	Bombay
44	224×224×3 uint8	Bombay
45	224×224×3 uint8	Bombay
46	224×224×3 uint8	Bombay
47	224×224×3 uint8	Bombay
48	224×224×3 uint8	Bombay
49	224×224×3 uint8	Bombay
50	224×224×3 uint8	Bombay

	input	response
51	224×224×3 uint8	Bombay
52	224×224×3 uint8	Bombay
53	224×224×3 uint8	Bombay
54	224×224×3 uint8	Bombay
55	224×224×3 uint8	Bombay
56	224×224×3 uint8	Bombay
57	224×224×3 uint8	Bombay
58	224×224×3 uint8	Bombay
59	224×224×3 uint8	Bombay
60	224×224×3 uint8	Bombay
61	224×224×3 uint8	British_Shorth...
62	224×224×3 uint8	British_Shorth...
63	224×224×3 uint8	British_Shorth...
64	224×224×3 uint8	British_Shorth...
65	224×224×3 uint8	British_Shorth...
66	224×224×3 uint8	British_Shorth...
67	224×224×3 uint8	British_Shorth...
68	224×224×3 uint8	British_Shorth...
69	224×224×3 uint8	British_Shorth...
70	224×224×3 uint8	British_Shorth...
71	224×224×3 uint8	British_Shorth...
72	224×224×3 uint8	British_Shorth...
73	224×224×3 uint8	British_Shorth...
74	224×224×3 uint8	British_Shorth...
75	224×224×3 uint8	British_Shorth...
76	224×224×3 uint8	British_Shorth...
77	224×224×3 uint8	British_Shorth...
78	224×224×3 uint8	British_Shorth...
79	224×224×3 uint8	British_Shorth...
80	224×224×3 uint8	British_Shorth...
81	224×224×3 uint8	Egyptian Mau
82	224×224×3 uint8	Egyptian Mau
83	224×224×3 uint8	Egyptian Mau

	input	response
84	224×224×3 uint8	Egyptian Mau
85	224×224×3 uint8	Egyptian Mau
86	224×224×3 uint8	Egyptian Mau
87	224×224×3 uint8	Egyptian Mau
88	224×224×3 uint8	Egyptian Mau
89	224×224×3 uint8	Egyptian Mau
90	224×224×3 uint8	Egyptian Mau
91	224×224×3 uint8	Egyptian Mau
92	224×224×3 uint8	Egyptian Mau
93	224×224×3 uint8	Egyptian Mau
94	224×224×3 uint8	Egyptian Mau
95	224×224×3 uint8	Egyptian Mau
96	224×224×3 uint8	Egyptian Mau
97	224×224×3 uint8	Egyptian Mau
98	224×224×3 uint8	Egyptian Mau
99	224×224×3 uint8	Egyptian Mau
100	224×224×3 uint8	Egyptian Mau
⋮		

Number of Samples in Validation Set

```
[num,~,ic] = unique(valset.response);

a_table = table(num,accumarray(ic,1))
```

a_table = 38×2 table

	num	Var2
1	Bengal	40
2	Birman	40
3	Bombay	40
4	British_Shorthair	40
5	Egyptian Mau	40
6	Maine_Coon	40
7	Persian	40
8	Ragdoll	40
9	Russian Blue	40

	num	Var2
10	Siamese	40
11	Sphynx	40
12	abyssinian	40
13	american bulldog	40
14	american pit bul...	40
15	american_pit_bul...	40
16	basset_hound	40
17	beagle	40
18	boxer	40
19	chihuahua	40
20	english cocker s...	40
21	english setter	40
22	german shorthaired	40
23	great pyrenees	40
24	havanese	40
25	japanese chin	40
26	keeshond	40
27	leonberger	40
28	miniature pinsche	40
29	newfoundland	40
30	pomeranian	40
31	pug	40
32	saint bernard	40
33	samoyed	40
34	scottish terrier	40
35	shiba inu	40
36	staffordshire bu...	38
37	wheaten terrier	40
38	yorkshire terrier	40

```
min(a_table.Var2)
```

```
ans = 38
```


Preparing Network for Training

Importing PreTrained Neural Network

Most pre-trained networks have been presented in a famous ImageNet Large Scale Visual Competition (ILSVRC). These networks are trained on 1000 categories each containing 1000 or so images. These networks are well trained in extracting different features of an image. So we transfer the weights of the initial layers of these networks and retrain on our set of images to classify accordingly.

```
net2 = googlenet
```

```
net2 =  
  DAGNetwork with properties:  
  
    Layers: [144x1 nnet.cnn.layer.Layer]  
    Connections: [170x2 table]  
    InputNames: {'data'}  
    OutputNames: {'output'}
```

(In case the network is not installed the command will generate a link to install the network and then it needs to be rerun)

Modifying the PreTrained Network Using Designer App

Once the network has been loaded open the app using the command where we will modify the network by changing the number of classes we are training the network on as well replace the last layer with a new one

For the classification problem at hand, we have 102 flower categories when we want our network to distinguish so we will replace

```
deepNetworkDesigner
```

Modifying the PreTrained Network Programmatically

```
lgraph = layerGraph(net2)
```

```
lgraph =  
  LayerGraph with properties:  
  
    Layers: [144x1 nnet.cnn.layer.Layer]  
    Connections: [170x2 table]  
    InputNames: {'data'}  
    OutputNames: {'output'}
```

```
lgraph = replaceLayer(lgraph, 'loss3-classifier', fullyConnectedLayer(38, "Name", 'new-fc', ...  
    'WeightLearnRateFactor', 10, "BiasLearnRateFactor", 10));
```

```
lgraph = replaceLayer(lgraph, 'output', classificationLayer("Name", 'newoutput'))
```

```
lgraph =
```

```
  LayerGraph with properties:
```

```
    Layers: [144x1 nnet.cnn.layer.Layer]
  Connections: [170x2 table]
    InputNames: {'data'}
    OutputNames: {'newoutput'}
```

```
layers = lgraph.Layers
```

```
layers =
```

```
  144x1 Layer array with layers:
```

1	'data'	Image Input	224x224x3 images with 'zerocenter' normalization
2	'conv1-7x7_s2'	Convolution	64 7x7x3 convolutions with stride [2 2] and padding [1 1]
3	'conv1-rel_7x7'	ReLU	ReLU
4	'pool1-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [1 1]
5	'pool1-norm1'	Cross Channel Normalization	cross channel normalization with 5 channels
6	'conv2-3x3_reduce'	Convolution	64 1x1x64 convolutions with stride [1 1] and padding [1 1]
7	'conv2-rel_3x3_reduce'	ReLU	ReLU
8	'conv2-3x3'	Convolution	192 3x3x64 convolutions with stride [1 1] and padding [1 1]
9	'conv2-rel_3x3'	ReLU	ReLU
10	'conv2-norm2'	Cross Channel Normalization	cross channel normalization with 5 channels
11	'pool2-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [1 1]
12	'inception_3a-1x1'	Convolution	64 1x1x192 convolutions with stride [1 1] and padding [1 1]
13	'inception_3a-rel_1x1'	ReLU	ReLU
14	'inception_3a-3x3_reduce'	Convolution	96 1x1x192 convolutions with stride [1 1] and padding [1 1]
15	'inception_3a-rel_3x3_reduce'	ReLU	ReLU
16	'inception_3a-3x3'	Convolution	128 3x3x96 convolutions with stride [1 1] and padding [1 1]
17	'inception_3a-rel_3x3'	ReLU	ReLU
18	'inception_3a-5x5_reduce'	Convolution	16 1x1x192 convolutions with stride [1 1] and padding [1 1]
19	'inception_3a-rel_5x5_reduce'	ReLU	ReLU
20	'inception_3a-5x5'	Convolution	32 5x5x16 convolutions with stride [1 1] and padding [1 1]
21	'inception_3a-rel_5x5'	ReLU	ReLU
22	'inception_3a-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [1 1]
23	'inception_3a-pool_proj'	Convolution	32 1x1x192 convolutions with stride [1 1] and padding [1 1]
24	'inception_3a-rel_pool_proj'	ReLU	ReLU
25	'inception_3a-output'	Depth concatenation	Depth concatenation of 4 inputs
26	'inception_3b-1x1'	Convolution	128 1x1x256 convolutions with stride [1 1] and padding [1 1]
27	'inception_3b-rel_1x1'	ReLU	ReLU
28	'inception_3b-3x3_reduce'	Convolution	128 1x1x256 convolutions with stride [1 1] and padding [1 1]
29	'inception_3b-rel_3x3_reduce'	ReLU	ReLU
30	'inception_3b-3x3'	Convolution	192 3x3x128 convolutions with stride [1 1] and padding [1 1]
31	'inception_3b-rel_3x3'	ReLU	ReLU
32	'inception_3b-5x5_reduce'	Convolution	32 1x1x256 convolutions with stride [1 1] and padding [1 1]
33	'inception_3b-rel_5x5_reduce'	ReLU	ReLU
34	'inception_3b-5x5'	Convolution	96 5x5x32 convolutions with stride [1 1] and padding [1 1]
35	'inception_3b-rel_5x5'	ReLU	ReLU
36	'inception_3b-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [1 1]
37	'inception_3b-pool_proj'	Convolution	64 1x1x256 convolutions with stride [1 1] and padding [1 1]
38	'inception_3b-rel_pool_proj'	ReLU	ReLU
39	'inception_3b-output'	Depth concatenation	Depth concatenation of 4 inputs
40	'pool3-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [1 1]
41	'inception_4a-1x1'	Convolution	192 1x1x480 convolutions with stride [1 1] and padding [1 1]
42	'inception_4a-rel_1x1'	ReLU	ReLU
43	'inception_4a-3x3_reduce'	Convolution	96 1x1x480 convolutions with stride [1 1] and padding [1 1]
44	'inception_4a-rel_3x3_reduce'	ReLU	ReLU
45	'inception_4a-3x3'	Convolution	208 3x3x96 convolutions with stride [1 1] and padding [1 1]
46	'inception_4a-rel_3x3'	ReLU	ReLU
47	'inception_4a-5x5_reduce'	Convolution	16 1x1x480 convolutions with stride [1 1] and padding [1 1]
48	'inception_4a-rel_5x5_reduce'	ReLU	ReLU

49	'inception_4a-5x5'	Convolution	48 5x5x16 convolutions with stride [1 1] and padding [2 2]
50	'inception_4a-rel_5x5'	ReLU	ReLU
51	'inception_4a-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
52	'inception_4a-pool_proj'	Convolution	64 1x1x480 convolutions with stride [1 1] and padding [0 0]
53	'inception_4a-rel_4a-pool_proj'	ReLU	ReLU
54	'inception_4a-output'	Depth concatenation	Depth concatenation of 4 inputs
55	'inception_4b-1x1'	Convolution	160 1x1x512 convolutions with stride [1 1] and padding [0 0]
56	'inception_4b-rel_1x1'	ReLU	ReLU
57	'inception_4b-3x3_reduce'	Convolution	112 1x1x512 convolutions with stride [1 1] and padding [0 0]
58	'inception_4b-rel_3x3_reduce'	ReLU	ReLU
59	'inception_4b-3x3'	Convolution	224 3x3x112 convolutions with stride [1 1] and padding [0 0]
60	'inception_4b-rel_3x3'	ReLU	ReLU
61	'inception_4b-5x5_reduce'	Convolution	24 1x1x512 convolutions with stride [1 1] and padding [0 0]
62	'inception_4b-rel_5x5_reduce'	ReLU	ReLU
63	'inception_4b-5x5'	Convolution	64 5x5x24 convolutions with stride [1 1] and padding [0 0]
64	'inception_4b-rel_5x5'	ReLU	ReLU
65	'inception_4b-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
66	'inception_4b-pool_proj'	Convolution	64 1x1x512 convolutions with stride [1 1] and padding [0 0]
67	'inception_4b-rel_4b-pool_proj'	ReLU	ReLU
68	'inception_4b-output'	Depth concatenation	Depth concatenation of 4 inputs
69	'inception_4c-1x1'	Convolution	128 1x1x512 convolutions with stride [1 1] and padding [0 0]
70	'inception_4c-rel_1x1'	ReLU	ReLU
71	'inception_4c-3x3_reduce'	Convolution	128 1x1x512 convolutions with stride [1 1] and padding [0 0]
72	'inception_4c-rel_3x3_reduce'	ReLU	ReLU
73	'inception_4c-3x3'	Convolution	256 3x3x128 convolutions with stride [1 1] and padding [0 0]
74	'inception_4c-rel_3x3'	ReLU	ReLU
75	'inception_4c-5x5_reduce'	Convolution	24 1x1x512 convolutions with stride [1 1] and padding [0 0]
76	'inception_4c-rel_5x5_reduce'	ReLU	ReLU
77	'inception_4c-5x5'	Convolution	64 5x5x24 convolutions with stride [1 1] and padding [0 0]
78	'inception_4c-rel_5x5'	ReLU	ReLU
79	'inception_4c-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
80	'inception_4c-pool_proj'	Convolution	64 1x1x512 convolutions with stride [1 1] and padding [0 0]
81	'inception_4c-rel_4c-pool_proj'	ReLU	ReLU
82	'inception_4c-output'	Depth concatenation	Depth concatenation of 4 inputs
83	'inception_4d-1x1'	Convolution	112 1x1x512 convolutions with stride [1 1] and padding [0 0]
84	'inception_4d-rel_1x1'	ReLU	ReLU
85	'inception_4d-3x3_reduce'	Convolution	144 1x1x512 convolutions with stride [1 1] and padding [0 0]
86	'inception_4d-rel_3x3_reduce'	ReLU	ReLU
87	'inception_4d-3x3'	Convolution	288 3x3x144 convolutions with stride [1 1] and padding [0 0]
88	'inception_4d-rel_3x3'	ReLU	ReLU
89	'inception_4d-5x5_reduce'	Convolution	32 1x1x512 convolutions with stride [1 1] and padding [0 0]
90	'inception_4d-rel_5x5_reduce'	ReLU	ReLU
91	'inception_4d-5x5'	Convolution	64 5x5x32 convolutions with stride [1 1] and padding [0 0]
92	'inception_4d-rel_5x5'	ReLU	ReLU
93	'inception_4d-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
94	'inception_4d-pool_proj'	Convolution	64 1x1x512 convolutions with stride [1 1] and padding [0 0]
95	'inception_4d-rel_4d-pool_proj'	ReLU	ReLU
96	'inception_4d-output'	Depth concatenation	Depth concatenation of 4 inputs
97	'inception_4e-1x1'	Convolution	256 1x1x528 convolutions with stride [1 1] and padding [0 0]
98	'inception_4e-rel_1x1'	ReLU	ReLU
99	'inception_4e-3x3_reduce'	Convolution	160 1x1x528 convolutions with stride [1 1] and padding [0 0]
100	'inception_4e-rel_3x3_reduce'	ReLU	ReLU
101	'inception_4e-3x3'	Convolution	320 3x3x160 convolutions with stride [1 1] and padding [0 0]
102	'inception_4e-rel_3x3'	ReLU	ReLU
103	'inception_4e-5x5_reduce'	Convolution	32 1x1x528 convolutions with stride [1 1] and padding [0 0]
104	'inception_4e-rel_5x5_reduce'	ReLU	ReLU
105	'inception_4e-5x5'	Convolution	128 5x5x32 convolutions with stride [1 1] and padding [0 0]
106	'inception_4e-rel_5x5'	ReLU	ReLU
107	'inception_4e-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
108	'inception_4e-pool_proj'	Convolution	128 1x1x528 convolutions with stride [1 1] and padding [0 0]
109	'inception_4e-rel_4e-pool_proj'	ReLU	ReLU
110	'inception_4e-output'	Depth concatenation	Depth concatenation of 4 inputs
111	'pool4-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
112	'inception_5a-1x1'	Convolution	256 1x1x832 convolutions with stride [1 1] and padding [0 0]

113	'inception_5a-relu_1x1'	ReLU	ReLU
114	'inception_5a-3x3_reduce'	Convolution	160 1×1×832 convolutions with stride [1 1]
115	'inception_5a-relu_3x3_reduce'	ReLU	ReLU
116	'inception_5a-3x3'	Convolution	320 3×3×160 convolutions with stride [1 1]
117	'inception_5a-relu_3x3'	ReLU	ReLU
118	'inception_5a-5x5_reduce'	Convolution	32 1×1×832 convolutions with stride [1 1]
119	'inception_5a-relu_5x5_reduce'	ReLU	ReLU
120	'inception_5a-5x5'	Convolution	128 5×5×32 convolutions with stride [1 1]
121	'inception_5a-relu_5x5'	ReLU	ReLU
122	'inception_5a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding
123	'inception_5a-pool_proj'	Convolution	128 1×1×832 convolutions with stride [1 1]
124	'inception_5a-relu_pool_proj'	ReLU	ReLU
125	'inception_5a-output'	Depth concatenation	Depth concatenation of 4 inputs
126	'inception_5b-1x1'	Convolution	384 1×1×832 convolutions with stride [1 1]
127	'inception_5b-relu_1x1'	ReLU	ReLU
128	'inception_5b-3x3_reduce'	Convolution	192 1×1×832 convolutions with stride [1 1]
129	'inception_5b-relu_3x3_reduce'	ReLU	ReLU
130	'inception_5b-3x3'	Convolution	384 3×3×192 convolutions with stride [1 1]
131	'inception_5b-relu_3x3'	ReLU	ReLU
132	'inception_5b-5x5_reduce'	Convolution	48 1×1×832 convolutions with stride [1 1]
133	'inception_5b-relu_5x5_reduce'	ReLU	ReLU
134	'inception_5b-5x5'	Convolution	128 5×5×48 convolutions with stride [1 1]
135	'inception_5b-relu_5x5'	ReLU	ReLU
136	'inception_5b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding
137	'inception_5b-pool_proj'	Convolution	128 1×1×832 convolutions with stride [1 1]
138	'inception_5b-relu_pool_proj'	ReLU	ReLU
139	'inception_5b-output'	Depth concatenation	Depth concatenation of 4 inputs
140	'pool5-7x7_s1'	2-D Global Average Pooling	2-D global average pooling
141	'pool5-drop_7x7_s1'	Dropout	40% dropout
142	'new-fc'	Fully Connected	38 fully connected layer
143	'prob'	Softmax	softmax
144	'newoutput'	Classification Output	crossentropyex

```
connections = lgraph.Connections
```

```
connections = 170×2 table
```

	Source	Destination
1	'data'	'conv1-7x7_s2'
2	'conv1-7x7_s2'	'conv1-relu_7x7'
3	'conv1-relu_7x7'	'pool1-3x3_s2'
4	'pool1-3x3_s2'	'pool1-norm1'
5	'pool1-norm1'	'conv2-3x3_reduce'
6	'conv2-3x3_reduce'	'conv2-relu_3x3_reduce'
7	'conv2-relu_3x3_reduce'	'conv2-3x3'
8	'conv2-3x3'	'conv2-relu_3x3'
9	'conv2-relu_3x3'	'conv2-norm2'
10	'conv2-norm2'	'pool2-3x3_s2'
11	'pool2-3x3_s2'	'inception_3a-1x1'
12	'pool2-3x3_s2'	'inception_3a-3x3_reduce'
13	'pool2-3x3_s2'	'inception_3a-5x5_reduce'
14	'pool2-3x3_s2'	'inception_3a-pool'

	Source	Destination
15	'inception_3a-1x1'	'inception_3a-relu_1x1'
16	'inception_3a-relu_1x1'	'inception_3a-output/in1'
17	'inception_3a-3x3_reduce'	'inception_3a-relu_3x3_...
18	'inception_3a-relu_3x3_...	'inception_3a-3x3'
19	'inception_3a-3x3'	'inception_3a-relu_3x3'
20	'inception_3a-relu_3x3'	'inception_3a-output/in2'
21	'inception_3a-5x5_reduce'	'inception_3a-relu_5x5_...
22	'inception_3a-relu_5x5_...	'inception_3a-5x5'
23	'inception_3a-5x5'	'inception_3a-relu_5x5'
24	'inception_3a-relu_5x5'	'inception_3a-output/in3'
25	'inception_3a-pool'	'inception_3a-pool_proj'
26	'inception_3a-pool_proj'	'inception_3a-relu_pool...
27	'inception_3a-relu_pool...	'inception_3a-output/in4'
28	'inception_3a-output'	'inception_3b-1x1'
29	'inception_3a-output'	'inception_3b-3x3_reduce'
30	'inception_3a-output'	'inception_3b-5x5_reduce'
31	'inception_3a-output'	'inception_3b-pool'
32	'inception_3b-1x1'	'inception_3b-relu_1x1'
33	'inception_3b-relu_1x1'	'inception_3b-output/in1'
34	'inception_3b-3x3_reduce'	'inception_3b-relu_3x3_...
35	'inception_3b-relu_3x3_...	'inception_3b-3x3'
36	'inception_3b-3x3'	'inception_3b-relu_3x3'
37	'inception_3b-relu_3x3'	'inception_3b-output/in2'
38	'inception_3b-5x5_reduce'	'inception_3b-relu_5x5_...
39	'inception_3b-relu_5x5_...	'inception_3b-5x5'
40	'inception_3b-5x5'	'inception_3b-relu_5x5'
41	'inception_3b-relu_5x5'	'inception_3b-output/in3'
42	'inception_3b-pool'	'inception_3b-pool_proj'
43	'inception_3b-pool_proj'	'inception_3b-relu_pool...
44	'inception_3b-relu_pool...	'inception_3b-output/in4'
45	'inception_3b-output'	'pool3-3x3_s2'
46	'pool3-3x3_s2'	'inception_4a-1x1'
47	'pool3-3x3_s2'	'inception_4a-3x3_reduce'

	Source	Destination
48	'pool3-3x3_s2'	'inception_4a-5x5_reduce'
49	'pool3-3x3_s2'	'inception_4a-pool'
50	'inception_4a-1x1'	'inception_4a-relu_1x1'
51	'inception_4a-relu_1x1'	'inception_4a-output/in1'
52	'inception_4a-3x3_reduce'	'inception_4a-relu_3x3_...
53	'inception_4a-relu_3x3_...	'inception_4a-3x3'
54	'inception_4a-3x3'	'inception_4a-relu_3x3'
55	'inception_4a-relu_3x3'	'inception_4a-output/in2'
56	'inception_4a-5x5_reduce'	'inception_4a-relu_5x5_...
57	'inception_4a-relu_5x5_...	'inception_4a-5x5'
58	'inception_4a-5x5'	'inception_4a-relu_5x5'
59	'inception_4a-relu_5x5'	'inception_4a-output/in3'
60	'inception_4a-pool'	'inception_4a-pool_proj'
61	'inception_4a-pool_proj'	'inception_4a-relu_pool...
62	'inception_4a-relu_pool...	'inception_4a-output/in4'
63	'inception_4a-output'	'inception_4b-1x1'
64	'inception_4a-output'	'inception_4b-3x3_reduce'
65	'inception_4a-output'	'inception_4b-5x5_reduce'
66	'inception_4a-output'	'inception_4b-pool'
67	'inception_4b-1x1'	'inception_4b-relu_1x1'
68	'inception_4b-relu_1x1'	'inception_4b-output/in1'
69	'inception_4b-3x3_reduce'	'inception_4b-relu_3x3_...
70	'inception_4b-relu_3x3_...	'inception_4b-3x3'
71	'inception_4b-3x3'	'inception_4b-relu_3x3'
72	'inception_4b-relu_3x3'	'inception_4b-output/in2'
73	'inception_4b-5x5_reduce'	'inception_4b-relu_5x5_...
74	'inception_4b-relu_5x5_...	'inception_4b-5x5'
75	'inception_4b-5x5'	'inception_4b-relu_5x5'
76	'inception_4b-relu_5x5'	'inception_4b-output/in3'
77	'inception_4b-pool'	'inception_4b-pool_proj'
78	'inception_4b-pool_proj'	'inception_4b-relu_pool...
79	'inception_4b-relu_pool...	'inception_4b-output/in4'
80	'inception_4b-output'	'inception_4c-1x1'

	Source	Destination
81	'inception_4b-output'	'inception_4c-3x3_reduce'
82	'inception_4b-output'	'inception_4c-5x5_reduce'
83	'inception_4b-output'	'inception_4c-pool'
84	'inception_4c-1x1'	'inception_4c-relu_1x1'
85	'inception_4c-relu_1x1'	'inception_4c-output/in1'
86	'inception_4c-3x3_reduce'	'inception_4c-relu_3x3_...
87	'inception_4c-relu_3x3_...	'inception_4c-3x3'
88	'inception_4c-3x3'	'inception_4c-relu_3x3'
89	'inception_4c-relu_3x3'	'inception_4c-output/in2'
90	'inception_4c-5x5_reduce'	'inception_4c-relu_5x5_...
91	'inception_4c-relu_5x5_...	'inception_4c-5x5'
92	'inception_4c-5x5'	'inception_4c-relu_5x5'
93	'inception_4c-relu_5x5'	'inception_4c-output/in3'
94	'inception_4c-pool'	'inception_4c-pool_proj'
95	'inception_4c-pool_proj'	'inception_4c-relu_pool_...
96	'inception_4c-relu_pool_...	'inception_4c-output/in4'
97	'inception_4c-output'	'inception_4d-1x1'
98	'inception_4c-output'	'inception_4d-3x3_reduce'
99	'inception_4c-output'	'inception_4d-5x5_reduce'
100	'inception_4c-output'	'inception_4d-pool'

⋮

```
layers(1:10) = freezeWeights(layers(1:10))
```

layers =

144x1 Layer array with layers:

1	'data'	Image Input	224x224x3 images with 'zerocenter' normalization
2	'conv1-7x7_s2'	Convolution	64 7x7x3 convolutions with stride [2 2] and padding [1 1]
3	'conv1-relu_7x7'	ReLU	ReLU
4	'pool1-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [1 1]
5	'pool1-norm1'	Cross Channel Normalization	cross channel normalization with 5 channels
6	'conv2-3x3_reduce'	Convolution	64 1x1x64 convolutions with stride [1 1] and padding [0 0]
7	'conv2-relu_3x3_reduce'	ReLU	ReLU
8	'conv2-3x3'	Convolution	192 3x3x64 convolutions with stride [1 1] and padding [0 0]
9	'conv2-relu_3x3'	ReLU	ReLU
10	'conv2-norm2'	Cross Channel Normalization	cross channel normalization with 5 channels
11	'pool2-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [1 1]
12	'inception_3a-1x1'	Convolution	64 1x1x192 convolutions with stride [1 1] and padding [0 0]
13	'inception_3a-relu_1x1'	ReLU	ReLU
14	'inception_3a-3x3_reduce'	Convolution	96 1x1x192 convolutions with stride [1 1] and padding [0 0]
15	'inception_3a-relu_3x3_reduce'	ReLU	ReLU
16	'inception_3a-3x3'	Convolution	128 3x3x96 convolutions with stride [1 1] and padding [0 0]
17	'inception_3a-relu_3x3'	ReLU	ReLU

18	'inception_3a-5x5_reduce'	Convolution	16 1x1x192 convolutions with stride [1 1] and padding [0 0]
19	'inception_3a-relu_5x5_reduce'	ReLU	ReLU
20	'inception_3a-5x5'	Convolution	32 5x5x16 convolutions with stride [1 1] and padding [0 0]
21	'inception_3a-relu_5x5'	ReLU	ReLU
22	'inception_3a-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
23	'inception_3a-pool_proj'	Convolution	32 1x1x192 convolutions with stride [1 1] and padding [0 0]
24	'inception_3a-relu_pool_proj'	ReLU	ReLU
25	'inception_3a-output'	Depth concatenation	Depth concatenation of 4 inputs
26	'inception_3b-1x1'	Convolution	128 1x1x256 convolutions with stride [1 1] and padding [0 0]
27	'inception_3b-relu_1x1'	ReLU	ReLU
28	'inception_3b-3x3_reduce'	Convolution	128 1x1x256 convolutions with stride [1 1] and padding [0 0]
29	'inception_3b-relu_3x3_reduce'	ReLU	ReLU
30	'inception_3b-3x3'	Convolution	192 3x3x128 convolutions with stride [1 1] and padding [0 0]
31	'inception_3b-relu_3x3'	ReLU	ReLU
32	'inception_3b-5x5_reduce'	Convolution	32 1x1x256 convolutions with stride [1 1] and padding [0 0]
33	'inception_3b-relu_5x5_reduce'	ReLU	ReLU
34	'inception_3b-5x5'	Convolution	96 5x5x32 convolutions with stride [1 1] and padding [0 0]
35	'inception_3b-relu_5x5'	ReLU	ReLU
36	'inception_3b-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
37	'inception_3b-pool_proj'	Convolution	64 1x1x256 convolutions with stride [1 1] and padding [0 0]
38	'inception_3b-relu_pool_proj'	ReLU	ReLU
39	'inception_3b-output'	Depth concatenation	Depth concatenation of 4 inputs
40	'pool3-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
41	'inception_4a-1x1'	Convolution	192 1x1x480 convolutions with stride [1 1] and padding [0 0]
42	'inception_4a-relu_1x1'	ReLU	ReLU
43	'inception_4a-3x3_reduce'	Convolution	96 1x1x480 convolutions with stride [1 1] and padding [0 0]
44	'inception_4a-relu_3x3_reduce'	ReLU	ReLU
45	'inception_4a-3x3'	Convolution	208 3x3x96 convolutions with stride [1 1] and padding [0 0]
46	'inception_4a-relu_3x3'	ReLU	ReLU
47	'inception_4a-5x5_reduce'	Convolution	16 1x1x480 convolutions with stride [1 1] and padding [0 0]
48	'inception_4a-relu_5x5_reduce'	ReLU	ReLU
49	'inception_4a-5x5'	Convolution	48 5x5x16 convolutions with stride [1 1] and padding [0 0]
50	'inception_4a-relu_5x5'	ReLU	ReLU
51	'inception_4a-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
52	'inception_4a-pool_proj'	Convolution	64 1x1x480 convolutions with stride [1 1] and padding [0 0]
53	'inception_4a-relu_pool_proj'	ReLU	ReLU
54	'inception_4a-output'	Depth concatenation	Depth concatenation of 4 inputs
55	'inception_4b-1x1'	Convolution	160 1x1x512 convolutions with stride [1 1] and padding [0 0]
56	'inception_4b-relu_1x1'	ReLU	ReLU
57	'inception_4b-3x3_reduce'	Convolution	112 1x1x512 convolutions with stride [1 1] and padding [0 0]
58	'inception_4b-relu_3x3_reduce'	ReLU	ReLU
59	'inception_4b-3x3'	Convolution	224 3x3x112 convolutions with stride [1 1] and padding [0 0]
60	'inception_4b-relu_3x3'	ReLU	ReLU
61	'inception_4b-5x5_reduce'	Convolution	24 1x1x512 convolutions with stride [1 1] and padding [0 0]
62	'inception_4b-relu_5x5_reduce'	ReLU	ReLU
63	'inception_4b-5x5'	Convolution	64 5x5x24 convolutions with stride [1 1] and padding [0 0]
64	'inception_4b-relu_5x5'	ReLU	ReLU
65	'inception_4b-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
66	'inception_4b-pool_proj'	Convolution	64 1x1x512 convolutions with stride [1 1] and padding [0 0]
67	'inception_4b-relu_pool_proj'	ReLU	ReLU
68	'inception_4b-output'	Depth concatenation	Depth concatenation of 4 inputs
69	'inception_4c-1x1'	Convolution	128 1x1x512 convolutions with stride [1 1] and padding [0 0]
70	'inception_4c-relu_1x1'	ReLU	ReLU
71	'inception_4c-3x3_reduce'	Convolution	128 1x1x512 convolutions with stride [1 1] and padding [0 0]
72	'inception_4c-relu_3x3_reduce'	ReLU	ReLU
73	'inception_4c-3x3'	Convolution	256 3x3x128 convolutions with stride [1 1] and padding [0 0]
74	'inception_4c-relu_3x3'	ReLU	ReLU
75	'inception_4c-5x5_reduce'	Convolution	24 1x1x512 convolutions with stride [1 1] and padding [0 0]
76	'inception_4c-relu_5x5_reduce'	ReLU	ReLU
77	'inception_4c-5x5'	Convolution	64 5x5x24 convolutions with stride [1 1] and padding [0 0]
78	'inception_4c-relu_5x5'	ReLU	ReLU
79	'inception_4c-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padding [0 0]
80	'inception_4c-pool_proj'	Convolution	64 1x1x512 convolutions with stride [1 1] and padding [0 0]
81	'inception_4c-relu_pool_proj'	ReLU	ReLU

82	'inception_4c-output'	Depth concatenation	Depth concatenation of 4 inputs
83	'inception_4d-1x1'	Convolution	112 1x1x512 convolutions with stride [1 1]
84	'inception_4d-relu_1x1'	ReLU	ReLU
85	'inception_4d-3x3_reduce'	Convolution	144 1x1x512 convolutions with stride [1 1]
86	'inception_4d-relu_3x3_reduce'	ReLU	ReLU
87	'inception_4d-3x3'	Convolution	288 3x3x144 convolutions with stride [1 1]
88	'inception_4d-relu_3x3'	ReLU	ReLU
89	'inception_4d-5x5_reduce'	Convolution	32 1x1x512 convolutions with stride [1 1] a
90	'inception_4d-relu_5x5_reduce'	ReLU	ReLU
91	'inception_4d-5x5'	Convolution	64 5x5x32 convolutions with stride [1 1] ar
92	'inception_4d-relu_5x5'	ReLU	ReLU
93	'inception_4d-pool'	Max Pooling	3x3 max pooling with stride [1 1] and padd
94	'inception_4d-pool_proj'	Convolution	64 1x1x512 convolutions with stride [1 1] a
95	'inception_4d-relu_pool_proj'	ReLU	ReLU
96	'inception_4d-output'	Depth concatenation	Depth concatenation of 4 inputs
97	'inception_4e-1x1'	Convolution	256 1x1x528 convolutions with stride [1 1]
98	'inception_4e-relu_1x1'	ReLU	ReLU
99	'inception_4e-3x3_reduce'	Convolution	160 1x1x528 convolutions with stride [1 1]
100	'inception_4e-relu_3x3_reduce'	ReLU	ReLU
101	'inception_4e-3x3'	Convolution	320 3x3x160 convolutions with stride [1 1]
102	'inception_4e-relu_3x3'	ReLU	ReLU
103	'inception_4e-5x5_reduce'	Convolution	32 1x1x528 convolutions with stride [1 1]
104	'inception_4e-relu_5x5_reduce'	ReLU	ReLU
105	'inception_4e-5x5'	Convolution	128 5x5x32 convolutions with stride [1 1]
106	'inception_4e-relu_5x5'	ReLU	ReLU
107	'inception_4e-pool'	Max Pooling	3x3 max pooling with stride [1 1] and pad
108	'inception_4e-pool_proj'	Convolution	128 1x1x528 convolutions with stride [1 1]
109	'inception_4e-relu_pool_proj'	ReLU	ReLU
110	'inception_4e-output'	Depth concatenation	Depth concatenation of 4 inputs
111	'pool4-3x3_s2'	Max Pooling	3x3 max pooling with stride [2 2] and pad
112	'inception_5a-1x1'	Convolution	256 1x1x832 convolutions with stride [1 1]
113	'inception_5a-relu_1x1'	ReLU	ReLU
114	'inception_5a-3x3_reduce'	Convolution	160 1x1x832 convolutions with stride [1 1]
115	'inception_5a-relu_3x3_reduce'	ReLU	ReLU
116	'inception_5a-3x3'	Convolution	320 3x3x160 convolutions with stride [1 1]
117	'inception_5a-relu_3x3'	ReLU	ReLU
118	'inception_5a-5x5_reduce'	Convolution	32 1x1x832 convolutions with stride [1 1]
119	'inception_5a-relu_5x5_reduce'	ReLU	ReLU
120	'inception_5a-5x5'	Convolution	128 5x5x32 convolutions with stride [1 1]
121	'inception_5a-relu_5x5'	ReLU	ReLU
122	'inception_5a-pool'	Max Pooling	3x3 max pooling with stride [1 1] and pad
123	'inception_5a-pool_proj'	Convolution	128 1x1x832 convolutions with stride [1 1]
124	'inception_5a-relu_pool_proj'	ReLU	ReLU
125	'inception_5a-output'	Depth concatenation	Depth concatenation of 4 inputs
126	'inception_5b-1x1'	Convolution	384 1x1x832 convolutions with stride [1 1]
127	'inception_5b-relu_1x1'	ReLU	ReLU
128	'inception_5b-3x3_reduce'	Convolution	192 1x1x832 convolutions with stride [1 1]
129	'inception_5b-relu_3x3_reduce'	ReLU	ReLU
130	'inception_5b-3x3'	Convolution	384 3x3x192 convolutions with stride [1 1]
131	'inception_5b-relu_3x3'	ReLU	ReLU
132	'inception_5b-5x5_reduce'	Convolution	48 1x1x832 convolutions with stride [1 1]
133	'inception_5b-relu_5x5_reduce'	ReLU	ReLU
134	'inception_5b-5x5'	Convolution	128 5x5x48 convolutions with stride [1 1]
135	'inception_5b-relu_5x5'	ReLU	ReLU
136	'inception_5b-pool'	Max Pooling	3x3 max pooling with stride [1 1] and pad
137	'inception_5b-pool_proj'	Convolution	128 1x1x832 convolutions with stride [1 1]
138	'inception_5b-relu_pool_proj'	ReLU	ReLU
139	'inception_5b-output'	Depth concatenation	Depth concatenation of 4 inputs
140	'pool5-7x7_s1'	2-D Global Average Pooling	2-D global average pooling
141	'pool5-drop_7x7_s1'	Dropout	40% dropout
142	'new-fc'	Fully Connected	38 fully connected layer
143	'prob'	Softmax	softmax
144	'newoutput'	Classification Output	crossentropyex

```
lgraph = createLgraphUsingConnections(layers,connections)
```

```
lgraph =  
  LayerGraph with properties:  
  
    Layers: [144x1 nnet.cnn.layer.Layer]  
    Connections: [170x2 table]  
    InputNames: {'data'}  
    OutputNames: {'newoutput'}
```

Training Deep Neural Network

Training Parameters

Once we have replaced the layers of networks and exported the modified network back, all that is left is setting up training parameters such as

1. Optimisation algorithm
2. Batch of images to train back to back for one iteration
3. Total number of Epochs
4. Validation Data
5. Learning rate

```
miniBatchSize = 100;  
  
options = trainingOptions('sgdm', ...  
    'MiniBatchSize',miniBatchSize, ...  
    'MaxEpochs',30, ...  
    'InitialLearnRate',3e-4, ...  
    'Shuffle','every-epoch', ...  
    'ValidationFrequency', 50,...  
    'ValidationData',valset, ...  
    'Verbose',true, ...  
    'Plots','training-progress')
```

```
options =  
  TrainingOptionsSGDM with properties:  
  
    Momentum: 0.9000  
    InitialLearnRate: 3.0000e-04  
    LearnRateSchedule: 'none'  
    LearnRateDropFactor: 0.1000  
    LearnRateDropPeriod: 10  
    L2Regularization: 1.0000e-04  
    GradientThresholdMethod: 'l2norm'  
    GradientThreshold: Inf  
    MaxEpochs: 30  
    MiniBatchSize: 100  
    Verbose: 1  
    VerboseFrequency: 50  
    ValidationData: [1518x2 table]  
    ValidationFrequency: 50  
    ValidationPatience: Inf  
    Shuffle: 'every-epoch'
```

```

CheckpointPath: ''
CheckpointFrequency: 1
CheckpointFrequencyUnit: 'epoch'
ExecutionEnvironment: 'auto'
WorkerLoad: []
OutputFcn: []
Plots: 'training-progress'
SequenceLength: 'longest'
SequencePaddingValue: 0
SequencePaddingDirection: 'right'
DispatchInBackground: 0
ResetInputNormalization: 1
BatchNormalizationStatistics: 'population'
OutputNetwork: 'last-iteration'

```

```

if canUseGPU
    executionEnvironment = "gpu";
    numberOfGPUs = gpuDeviceCount("available");
    pool = parpool(numberOfGPUs);
else
    executionEnvironment = "cpu";
    pool = parpool;
end

```

Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 1).

Train the network

It all comes together now, the data and training options we have arranged previously.

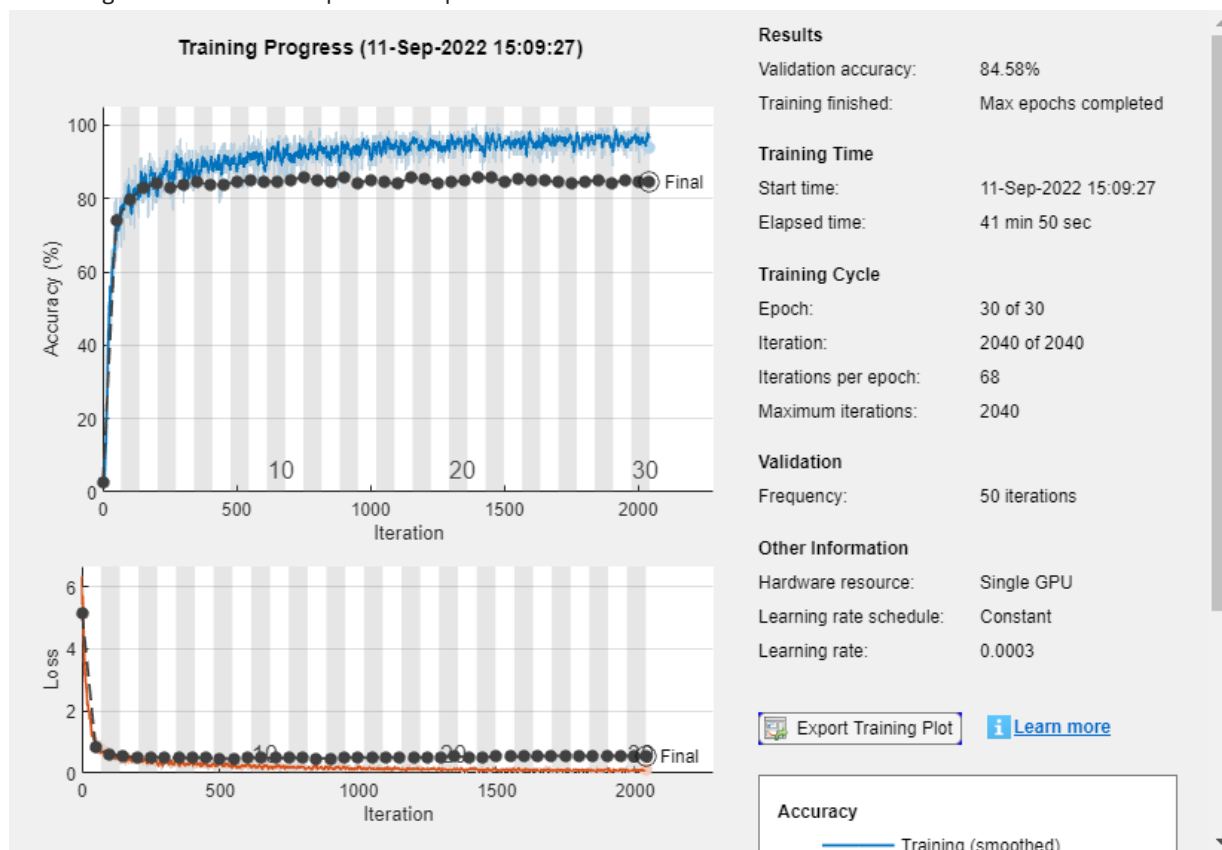
```
FlowerNet = trainNetwork(auds,lgraph,options)
```

Training on single GPU.
Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:26	1.00%	2.64%	6.3407	5.1372	0.001
1	50	00:01:26	78.00%	74.04%	0.7579	0.8267	0.001
2	100	00:02:22	77.00%	79.71%	0.6186	0.6175	0.001
3	150	00:03:20	86.00%	82.87%	0.4608	0.5370	0.001
3	200	00:04:18	80.00%	83.93%	0.5806	0.5161	0.001
4	250	00:05:13	88.00%	83.00%	0.3127	0.5172	0.001
5	300	00:06:09	88.00%	83.53%	0.3621	0.5105	0.001
6	350	00:07:05	86.00%	84.45%	0.3129	0.4840	0.001
6	400	00:08:03	93.00%	83.79%	0.2136	0.4935	0.001
7	450	00:09:03	93.00%	83.66%	0.2448	0.4995	0.001
8	500	00:10:03	85.00%	84.58%	0.3881	0.4615	0.001
9	550	00:11:02	89.00%	84.98%	0.2768	0.4755	0.001
9	600	00:12:02	90.00%	84.72%	0.2426	0.4970	0.001
10	650	00:13:05	94.00%	84.45%	0.1356	0.4874	0.001
11	700	00:14:07	91.00%	85.11%	0.2008	0.4819	0.001
12	750	00:15:09	90.00%	85.77%	0.1733	0.4838	0.001
12	800	00:16:09	95.00%	84.85%	0.1688	0.4932	0.001
13	850	00:17:09	89.00%	84.72%	0.2701	0.4722	0.001
14	900	00:18:07	97.00%	85.77%	0.1136	0.4752	0.001
14	950	00:19:08	92.00%	84.32%	0.2117	0.4987	0.001
15	1000	00:20:11	91.00%	84.98%	0.2310	0.5036	0.001

16	1050	00:21:18	96.00%	84.45%	0.1195	0.4960	0.00
17	1100	00:22:23	96.00%	83.93%	0.0910	0.5159	0.00
17	1150	00:23:25	96.00%	85.77%	0.1294	0.4906	0.00
18	1200	00:24:36	96.00%	85.38%	0.0808	0.5055	0.00
19	1250	00:25:42	94.00%	84.32%	0.1034	0.5098	0.00
20	1300	00:26:42	96.00%	84.45%	0.0942	0.5224	0.00
20	1350	00:27:37	94.00%	84.85%	0.1588	0.5427	0.00
21	1400	00:28:31	95.00%	85.64%	0.1466	0.5239	0.00
22	1450	00:29:26	97.00%	85.64%	0.0854	0.5201	0.00
23	1500	00:30:20	96.00%	84.58%	0.1030	0.5355	0.00
23	1550	00:31:15	96.00%	85.24%	0.0923	0.5320	0.00
24	1600	00:32:10	97.00%	84.98%	0.0977	0.5312	0.00
25	1650	00:33:10	96.00%	84.85%	0.0673	0.5449	0.00
25	1700	00:34:06	97.00%	84.45%	0.0803	0.5603	0.00
26	1750	00:35:02	96.00%	84.06%	0.0814	0.5554	0.00
27	1800	00:35:58	97.00%	84.72%	0.0890	0.5501	0.00
28	1850	00:38:17	98.00%	84.98%	0.0700	0.5536	0.00
28	1900	00:39:13	97.00%	84.32%	0.0647	0.5546	0.00
29	1950	00:40:08	92.00%	84.85%	0.1578	0.5541	0.00
30	2000	00:41:02	97.00%	84.58%	0.0603	0.5625	0.00
30	2040	00:41:46	94.00%	84.58%	0.1181	0.5683	0.00

Training finished: Max epochs completed.



FlowerNet =

DAGNetwork with properties:

```

Layers: [144x1 nnet.cnn.layer.Layer]
Connections: [170x2 table]
InputNames: {'data'}
OutputNames: {'newoutput'}

```

Evaluate the network

Before deploying the network we will estimate the accuracy of this network by testing the image data set that we separated earlier.

```
[predicted,scores] = classify(FlowerNet,valset);
tableofscores = [table(predicted) array2table(scores,"VariableNames",categories(valset.response
```

```
tableofscores = 1518x39 table
```

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
1	Bengal	0.9987	0	0	0	0.0008
2	Bengal	0.9998	0	0	0	0.0002
3	Bengal	0.9736	0	0	0	0.0264
4	Bengal	0.9861	0	0	0	0.0139
5	Bengal	0.9907	0	0	0	0.0091
6	Bengal	0.9998	0	0	0	0.0002
7	Bengal	0.9647	0	0	0	0.0352
8	Egyptian Mau	0.2645	0	0	0	0.7355
9	Bengal	0.9755	0	0.0002	0.0001	0.0200
10	Bengal	0.9996	0	0	0	0.0004
11	Bengal	0.9981	0	0	0	0.0019
12	Bengal	0.9591	0	0	0	0.0408
13	Bengal	0.6881	0	0	0	0.2784
14	Bengal	0.9622	0	0	0	0.0378
15	Bengal	0.9189	0	0	0	0.0811
16	Bengal	0.9989	0	0	0	0.0011
17	Egyptian Mau	0.3803	0	0	0.0001	0.6195
18	Bengal	0.8350	0	0	0	0.1396
19	Bengal	0.9964	0	0	0	0.0036
20	Bengal	0.9842	0	0	0	0.0156
21	Birman	0	0.9975	0	0	0
22	Birman	0	0.9978	0	0	0
23	Birman	0	0.9867	0	0	0
24	Birman	0	0.9944	0	0	0
25	Birman	0	0.8096	0	0	0
26	Birman	0	1	0	0	0
27	Birman	0	0.9988	0	0	0

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
28	Birman	0	0.8076	0	0	0
29	Birman	0	0.9999	0	0	0
30	Birman	0	1	0	0	0
31	Birman	0	1	0	0	0
32	Ragdoll	0	0.0659	0	0	0
33	Birman	0	0.9975	0	0	0
34	Birman	0.0003	0.3568	0.0006	0	0.0001
35	Birman	0	0.8719	0	0	0
36	Ragdoll	0	0.0088	0	0	0
37	Ragdoll	0	0.1001	0	0	0
38	Birman	0	0.9898	0	0	0
39	Birman	0	0.9981	0	0	0
40	pomeranian	0	0.0705	0	0	0
41	Bombay	0	0	0.9993	0.0006	0
42	Bombay	0	0	0.9985	0.0007	0
43	Bombay	0	0	0.9998	0	0
44	Bombay	0	0	1	0	0
45	Bombay	0	0	0.9999	0	0
46	Bombay	0	0	0.9910	0	0
47	Bombay	0.0003	0	0.9991	0.0002	0
48	Bombay	0	0	0.9999	0	0
49	Bombay	0	0	0.9914	0	0
50	Bombay	0	0	0.9944	0.0055	0
51	Bombay	0	0	0.9995	0.0001	0
52	Bengal	0.6300	0	0.3215	0.0029	0.0001
53	Bombay	0.0002	0	0.9918	0.0001	0
54	Bombay	0	0	1	0	0
55	Bombay	0	0	0.9998	0	0
56	Bombay	0	0	0.9999	0	0
57	Bombay	0.0008	0.0005	0.9573	0.0002	0
58	Bombay	0	0	1	0	0
59	Bombay	0	0	0.9954	0.0013	0
60	Bombay	0	0	1	0	0

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
61	Russian Blue	0.0902	0	0.0007	0.0317	0.0690
62	British_Shorthair	0.0021	0	0	0.6499	0.3459
63	British_Shorthair	0.3572	0	0	0.6258	0.0002
64	British_Shorthair	0	0	0.0001	0.9998	0
65	Bombay	0.0008	0	0.9955	0.0002	0
66	British_Shorthair	0	0	0.0018	0.9975	0
67	British_Shorthair	0	0	0	0.9602	0
68	Russian Blue	0	0	0.1266	0.0021	0
69	British_Shorthair	0.0001	0	0	0.8755	0.0001
70	British_Shorthair	0	0	0	0.9990	0
71	British_Shorthair	0	0	0	0.6241	0
72	British_Shorthair	0	0	0	1	0
73	British_Shorthair	0	0	0	1	0
74	British_Shorthair	0	0	0.0011	0.9923	0
75	British_Shorthair	0	0	0	0.9967	0
76	British_Shorthair	0	0	0	0.9999	0.0001
77	British_Shorthair	0	0	0	0.9999	0
78	British_Shorthair	0	0	0	0.9999	0
79	British_Shorthair	0	0	0	1	0
80	British_Shorthair	0	0	0.0010	0.9988	0
81	Egyptian Mau	0.0001	0	0	0	0.9999
82	Egyptian Mau	0	0	0	0	1
83	Egyptian Mau	0.0094	0	0	0	0.9906
84	Bombay	0.0305	0	0.8976	0	0.0038
85	Egyptian Mau	0.0338	0	0	0	0.9662
86	Egyptian Mau	0.0263	0	0	0	0.9737
87	Egyptian Mau	0.0006	0	0	0	0.9991
88	Russian Blue	0.2489	0	0.0003	0.0060	0.0298
89	Egyptian Mau	0.0045	0	0	0	0.9955
90	Egyptian Mau	0.0005	0	0	0	0.9995
91	Egyptian Mau	0.2115	0	0	0	0.5365
92	Bengal	0.9152	0	0	0	0.0820
93	Egyptian Mau	0.1521	0	0	0	0.8452

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
94	Egyptian Mau	0.0023	0	0	0	0.9976
95	Bengal	0.5036	0	0	0	0.4938
96	Egyptian Mau	0.0001	0	0	0	0.9999
97	Egyptian Mau	0.0012	0	0	0	0.9988
98	Egyptian Mau	0.0115	0	0	0	0.9884
99	Bengal	0.5941	0	0	0	0.4056
100	Egyptian Mau	0.0394	0	0	0	0.9606

⋮

```
nnz(predicted~=valset.response)/length(predicted)
```

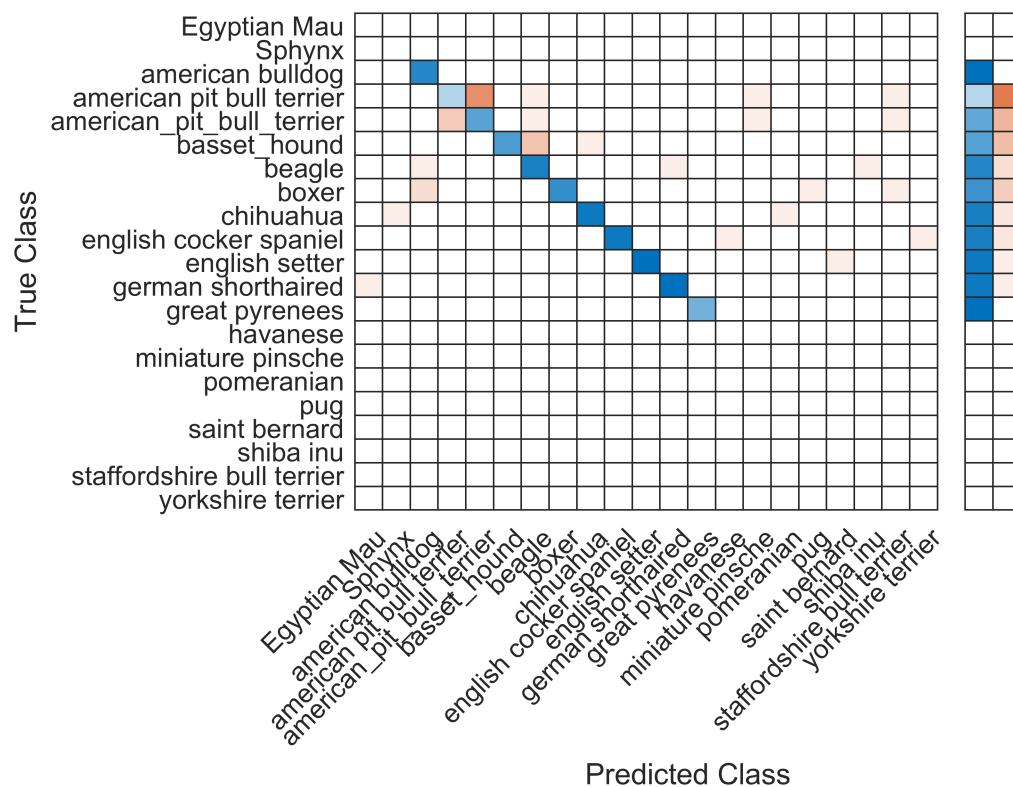
```
ans = 0.1542
```

Confusion Chart

Confusion chart can help in those categories which network is not able to classify properly so we can check our data and retrain the network with improved data to see if it rectifies the issue.

```
%cm2 = confusionchart(string(valset.response(245:450)),string(predicted(245:450)),"RowSummary",
```

```
cm2 = confusionchart(string(valset.response(245:450)),string(predicted(245:450)),"RowSummary",
```

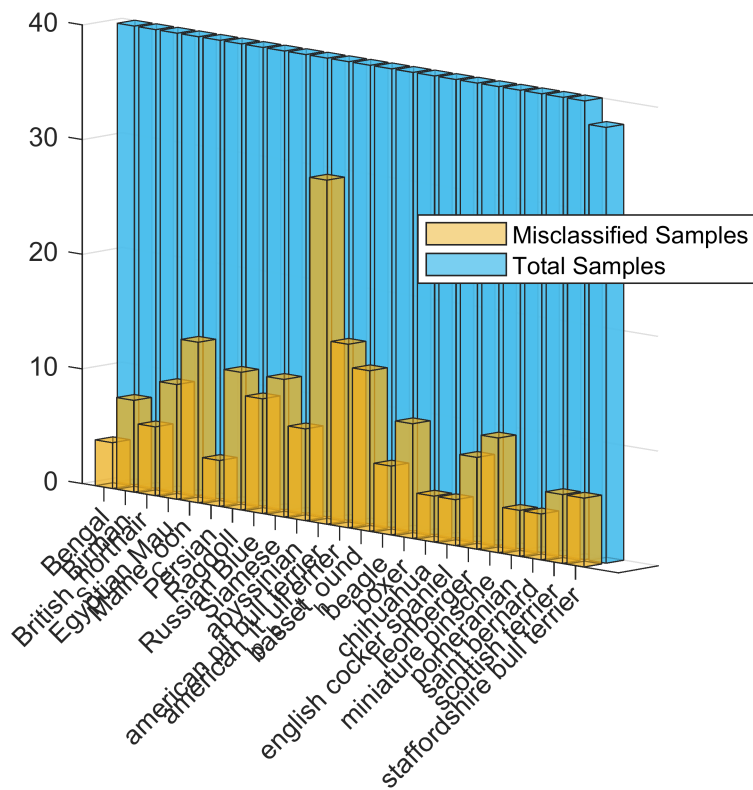



```
cm2 =
  ConfusionMatrixChart with properties:

    NormalizedValues: [21x21 double]
    ClassLabels: [21x1 string]

  Show all properties
```

```
mostmisclassified(valset,predicted)
```



Investigating Network Layers

While FlowerNet has many layers, I have selected an inception block that has convolution and pooling layers.

Using Activations

```
layer = "fc";
ind = find(valset.response == "abyssinian");
N = ind(6);
imshow(valset.input{N})
```



In convolution neural networks we can find out the activations of a particular convolutional or pooling layers and understand how it is contributing to the classification. Activations give a visual pattern as a result of a filter applied to that image

```
acs1 = activations(FlowerNet, valset(N,:), layer);
```

```
Error using DAGNetwork/activations>iValidateLayerName  
Layer 'fc' does not exist.
```

```
Error in DAGNetwork/activations>iGetSourceInformation (line 282)  
iValidateLayerName( startLayerName, layers );
```

```
Error in DAGNetwork/activations (line 240)  
iGetSourceInformation(layerOut, this.Layers);
```

```
size(ac1)  
imshow(imresize(imtile(rescale(ac1)), [1000 1000]))
```

Besides giving us insight on the network, activations can help us find nearest neighbor of the sample and we can determine how that particular layer is working. It can give same colour neighbours, same background neighbours, or same shape neighbours and other features like that. It would change from layer to layer. However, there is little control on what each channel in a layer learns.

```
layer = "fc";  
ind = find(valset.response == "abyssinian");  
N = ind(6);  
acs = activations(FlowerNet, aucs, layer, "MiniBatchSize", 1, "OutputAs", "rows");
```

```
Error using DAGNetwork/activations>iValidateLayerName  
Layer 'fc' does not exist.
```

```
Error in DAGNetwork/activations>iGetSourceInformation (line 282)  
iValidateLayerName( startLayerName, layers );
```

```
Error in DAGNetwork/activations (line 240)
    iGetSourceInformation(layerOut, this.Layers);
```

```
acs1 = activations(FlowerNet,valset(N,:),layer,"OutputAs","rows");
```

```
[idx, d] = knnsearch(acs, acs1, 'K', 5,"NSMethod","kdtree"); %machine learning algorithm of t
```

```
Unrecognized function or variable 'acs'.
```

```
%idx(1) = [];
searchresults = readByIndex(auds,idx);
```

```
montage([valset.input(N); searchresults.input])
```

```
Unable to resolve the name 'searchresults.input'.
```

```
labelmontage(valset,searchresults,N)
```

Using Deep Dream Image

DeepDreamImage feature helps understand us interpreting a networks visualisation of an image. Each layer has it's own way of looking at images in abstract sense

```
%DD = deepDreamImage(FlowerNet,'fc',[20 23 38],"Verbose",false,"NumIterations",10,"ExecutionEnv
DD = deepDreamImage(FlowerNet,'fc',[20 23 38],"Verbose",false,"NumIterations",10); %20-camellia
```

```
Error using deepDreamImage
Layer 'fc' does not exist.
```

```
imshow(imtile(DD(:,:,,1:3),"BackgroundColor", [0 0.2 0.5]))
axis on
text(100,35,'pug','FontSize',35); text(500,35,'boxer','FontSize',35); text(100,465,'japanese ch
```

Deploying Network

We are going to implement this network in image batch processor that uploads images and classifies using the network we trained

```
imageBatchProcessor
```

```
function lgraph = createLgraphUsingConnections(layers,connections)

lgraph = layerGraph();
for i = 1:numel(layers)
    lgraph = addLayers(lgraph,layers(i));
end
```

```

for c = 1:size(connections,1)
    lgraph = connectLayers(lgraph,connections.Source{c},connections.Destination{c});
end

end

```

```

function layers = freezeWeights(layers)

```

```

for ii = 1:size(layers,1)
    props = properties(layers(ii));
    for p = 1:numel(props)
        propName = props{p};
        if ~isempty(regexpi(propName, 'LearnRateFactor$', 'once'))
            layers(ii).(propName) = 0;
        end
    end
end
end

```

```

end

```

```

function labelmontage(valset,searchresults,N)

```

```

if height(searchresults) == 5
text(100,35,string(valset.response(N)),"FontSize",15); text(400,35,searchresults.response(1),"Color",[0.3 0.3 0.3],"FontSize",15);
text(740,35,searchresults.response(2),"Color",[0.3 0.3 0.3],"FontSize",15);
text(100,365,searchresults.response(3),"Color",[0.3 0.3 0.3],"FontSize",20); text(400,365,searchresults.response(4),"Color",[0.3 0.3 0.3],"FontSize",20);
text(780,365,searchresults.response(5),"Color",[0.3 0.3 0.3],"FontSize",20)

```

```

end

```

```

end

```

```

function mostmisclassified(valset,predicted)

```

```

%
v = valset.response(predicted~=valset.response);
[num,~,ic] = unique(v);
a_table = table(num,accumarray(ic,1));
[c,~,id] = unique(valset.response(ismember(valset.response,a_table.num(a_table.Var2>2))));
ac = table(c,accumarray(id,1));
b = bar3([a_table.Var2(a_table.Var2>2) ac.Var2]);
a = b.Parent;
a.XGrid = 'off';
a.YGrid = 'off';
a.XTick = [];
b(1).FaceColor = [0.93,0.69,0.13];
b(1).FaceAlpha = 0.5;

b(2).FaceColor = [0.3 0.75 0.93];
b(2).FaceAlpha = 0.75;
yticks(1:length(c))

```

```
yticklabels(string(a_table.num(a_table.Var2>2)));  
ytickangle(45);  
  
view([-45.4 9.7])  
  
legend('Misclassified Samples','Total Samples',"Location","best")  
end
```