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 sutiable 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 augmenation (modification) to images seemlessly 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: {6831×1 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 = 38×2 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 = 1518×2 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 accodingly.

```
net2 = googlenet
```

```
net2 =
  DAGNetwork with properties:

         Layers: [144×1 nnet.cnn.layer.Layer]
    Connections: [170×2 table]
     InputNames: {'data'}
    OutputNames: {'output'}
```

(In case the network is not installed the commad 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 clasification 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: [144×1 nnet.cnn.layer.Layer]
    Connections: [170×2 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: [144×1 nnet.cnn.layer.Layer]
    Connections: [170×2 table]
     InputNames: {'data'}
    OutputNames: {'newoutput'}
```

```
layers = lgraph.Layers
```

```
layers =
  144×1 Layer array with layers:

     1   'data'                          Image Input                 224×224×3 images with 'zerocenter' normaliza
     2   'conv1-7x7_s2'                  Convolution                 64 7×7×3 convolutions with stride [2  2] and
     3   'conv1-relu_7x7'                ReLU                        ReLU
     4   'pool1-3x3_s2'                  Max Pooling                 3×3 max pooling with stride [2  2] and paddi
     5   'pool1-norm1'                   Cross Channel Normalization cross channel normalization with 5 channels
     6   'conv2-3x3_reduce'              Convolution                 64 1×1×64 convolutions with stride [1  1] a
     7   'conv2-relu_3x3_reduce'         ReLU                        ReLU
     8   'conv2-3x3'                     Convolution                 192 3×3×64 convolutions with stride [1  1] a
     9   'conv2-relu_3x3'                ReLU                        ReLU
    10   'conv2-norm2'                   Cross Channel Normalization cross channel normalization with 5 channels
    11   'pool2-3x3_s2'                  Max Pooling                 3×3 max pooling with stride [2  2] and paddi
    12   'inception_3a-1x1'              Convolution                 64 1×1×192 convolutions with stride [1  1] a
    13   'inception_3a-relu_1x1'         ReLU                        ReLU
    14   'inception_3a-3x3_reduce'       Convolution                 96 1×1×192 convolutions with stride [1  1] a
    15   'inception_3a-relu_3x3_reduce'  ReLU                        ReLU
    16   'inception_3a-3x3'              Convolution                 128 3×3×96 convolutions with stride [1  1] a
    17   'inception_3a-relu_3x3'         ReLU                        ReLU
    18   'inception_3a-5x5_reduce'       Convolution                 16 1×1×192 convolutions with stride [1  1] a
    19   'inception_3a-relu_5x5_reduce'  ReLU                        ReLU
    20   'inception_3a-5x5'              Convolution                 32 5×5×16 convolutions with stride [1  1] a
    21   'inception_3a-relu_5x5'         ReLU                        ReLU
    22   'inception_3a-pool'             Max Pooling                 3×3 max pooling with stride [1  1] and paddi
    23   'inception_3a-pool_proj'        Convolution                 32 1×1×192 convolutions with stride [1  1] a
    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 1×1×256 convolutions with stride [1  1]
    27   'inception_3b-relu_1x1'         ReLU                        ReLU
    28   'inception_3b-3x3_reduce'       Convolution                 128 1×1×256 convolutions with stride [1  1]
    29   'inception_3b-relu_3x3_reduce'  ReLU                        ReLU
    30   'inception_3b-3x3'              Convolution                 192 3×3×128 convolutions with stride [1  1]
    31   'inception_3b-relu_3x3'         ReLU                        ReLU
    32   'inception_3b-5x5_reduce'       Convolution                 32 1×1×256 convolutions with stride [1  1] a
    33   'inception_3b-relu_5x5_reduce'  ReLU                        ReLU
    34   'inception_3b-5x5'              Convolution                 96 5×5×32 convolutions with stride [1  1] a
    35   'inception_3b-relu_5x5'         ReLU                        ReLU
    36   'inception_3b-pool'             Max Pooling                 3×3 max pooling with stride [1  1] and paddi
    37   'inception_3b-pool_proj'        Convolution                 64 1×1×256 convolutions with stride [1  1] a
    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                 3×3 max pooling with stride [2  2] and paddi
    41   'inception_4a-1x1'              Convolution                 192 1×1×480 convolutions with stride [1  1]
    42   'inception_4a-relu_1x1'         ReLU                        ReLU
    43   'inception_4a-3x3_reduce'       Convolution                 96 1×1×480 convolutions with stride [1  1] a
    44   'inception_4a-relu_3x3_reduce'  ReLU                        ReLU
    45   'inception_4a-3x3'              Convolution                 208 3×3×96 convolutions with stride [1  1] a
    46   'inception_4a-relu_3x3'         ReLU                        ReLU
    47   'inception_4a-5x5_reduce'       Convolution                 16 1×1×480 convolutions with stride [1  1] a
    48   'inception_4a-relu_5x5_reduce'  ReLU                        ReLU
```

49	'inception_4a-5x5'	Convolution	48 5×5×16 convolutions with stride [1 1] ar
50	'inception_4a-relu_5x5'	ReLU	ReLU
51	'inception_4a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
52	'inception_4a-pool_proj'	Convolution	64 1×1×480 convolutions with stride [1 1] a
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 1×1×512 convolutions with stride [1 1]
56	'inception_4b-relu_1x1'	ReLU	ReLU
57	'inception_4b-3x3_reduce'	Convolution	112 1×1×512 convolutions with stride [1 1]
58	'inception_4b-relu_3x3_reduce'	ReLU	ReLU
59	'inception_4b-3x3'	Convolution	224 3×3×112 convolutions with stride [1 1]
60	'inception_4b-relu_3x3'	ReLU	ReLU
61	'inception_4b-5x5_reduce'	Convolution	24 1×1×512 convolutions with stride [1 1] a
62	'inception_4b-relu_5x5_reduce'	ReLU	ReLU
63	'inception_4b-5x5'	Convolution	64 5×5×24 convolutions with stride [1 1] ar
64	'inception_4b-relu_5x5'	ReLU	ReLU
65	'inception_4b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
66	'inception_4b-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] a
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 1×1×512 convolutions with stride [1 1]
70	'inception_4c-relu_1x1'	ReLU	ReLU
71	'inception_4c-3x3_reduce'	Convolution	128 1×1×512 convolutions with stride [1 1]
72	'inception_4c-relu_3x3_reduce'	ReLU	ReLU
73	'inception_4c-3x3'	Convolution	256 3×3×128 convolutions with stride [1 1]
74	'inception_4c-relu_3x3'	ReLU	ReLU
75	'inception_4c-5x5_reduce'	Convolution	24 1×1×512 convolutions with stride [1 1] a
76	'inception_4c-relu_5x5_reduce'	ReLU	ReLU
77	'inception_4c-5x5'	Convolution	64 5×5×24 convolutions with stride [1 1] ar
78	'inception_4c-relu_5x5'	ReLU	ReLU
79	'inception_4c-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
80	'inception_4c-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] a
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 1×1×512 convolutions with stride [1 1]
84	'inception_4d-relu_1x1'	ReLU	ReLU
85	'inception_4d-3x3_reduce'	Convolution	144 1×1×512 convolutions with stride [1 1]
86	'inception_4d-relu_3x3_reduce'	ReLU	ReLU
87	'inception_4d-3x3'	Convolution	288 3×3×144 convolutions with stride [1 1]
88	'inception_4d-relu_3x3'	ReLU	ReLU
89	'inception_4d-5x5_reduce'	Convolution	32 1×1×512 convolutions with stride [1 1] a
90	'inception_4d-relu_5x5_reduce'	ReLU	ReLU
91	'inception_4d-5x5'	Convolution	64 5×5×32 convolutions with stride [1 1] ar
92	'inception_4d-relu_5x5'	ReLU	ReLU
93	'inception_4d-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
94	'inception_4d-pool_proj'	Convolution	64 1×1×512 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 1×1×528 convolutions with stride [1 1]
98	'inception_4e-relu_1x1'	ReLU	ReLU
99	'inception_4e-3x3_reduce'	Convolution	160 1×1×528 convolutions with stride [1 1]
100	'inception_4e-relu_3x3_reduce'	ReLU	ReLU
101	'inception_4e-3x3'	Convolution	320 3×3×160 convolutions with stride [1 1]
102	'inception_4e-relu_3x3'	ReLU	ReLU
103	'inception_4e-5x5_reduce'	Convolution	32 1×1×528 convolutions with stride [1 1]
104	'inception_4e-relu_5x5_reduce'	ReLU	ReLU
105	'inception_4e-5x5'	Convolution	128 5×5×32 convolutions with stride [1 1]
106	'inception_4e-relu_5x5'	ReLU	ReLU
107	'inception_4e-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padd
108	'inception_4e-pool_proj'	Convolution	128 1×1×528 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	3×3 max pooling with stride [2 2] and padd
112	'inception_5a-1x1'	Convolution	256 1×1×832 convolutions with stride [1 1]

```
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 padd
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 padd
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 =
  144×1 Layer array with layers:

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

18	'inception_3a-5x5_reduce'	Convolution	16 1×1×192 convolutions with stride [1 1] a
19	'inception_3a-relu_5x5_reduce'	ReLU	ReLU
20	'inception_3a-5x5'	Convolution	32 5×5×16 convolutions with stride [1 1] ar
21	'inception_3a-relu_5x5'	ReLU	ReLU
22	'inception_3a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
23	'inception_3a-pool_proj'	Convolution	32 1×1×192 convolutions with stride [1 1] a
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 1×1×256 convolutions with stride [1 1]
27	'inception_3b-relu_1x1'	ReLU	ReLU
28	'inception_3b-3x3_reduce'	Convolution	128 1×1×256 convolutions with stride [1 1]
29	'inception_3b-relu_3x3_reduce'	ReLU	ReLU
30	'inception_3b-3x3'	Convolution	192 3×3×128 convolutions with stride [1 1]
31	'inception_3b-relu_3x3'	ReLU	ReLU
32	'inception_3b-5x5_reduce'	Convolution	32 1×1×256 convolutions with stride [1 1] a
33	'inception_3b-relu_5x5_reduce'	ReLU	ReLU
34	'inception_3b-5x5'	Convolution	96 5×5×32 convolutions with stride [1 1] ar
35	'inception_3b-relu_5x5'	ReLU	ReLU
36	'inception_3b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
37	'inception_3b-pool_proj'	Convolution	64 1×1×256 convolutions with stride [1 1] a
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	3×3 max pooling with stride [2 2] and paddi
41	'inception_4a-1x1'	Convolution	192 1×1×480 convolutions with stride [1 1]
42	'inception_4a-relu_1x1'	ReLU	ReLU
43	'inception_4a-3x3_reduce'	Convolution	96 1×1×480 convolutions with stride [1 1] a
44	'inception_4a-relu_3x3_reduce'	ReLU	ReLU
45	'inception_4a-3x3'	Convolution	208 3×3×96 convolutions with stride [1 1] a
46	'inception_4a-relu_3x3'	ReLU	ReLU
47	'inception_4a-5x5_reduce'	Convolution	16 1×1×480 convolutions with stride [1 1] a
48	'inception_4a-relu_5x5_reduce'	ReLU	ReLU
49	'inception_4a-5x5'	Convolution	48 5×5×16 convolutions with stride [1 1] ar
50	'inception_4a-relu_5x5'	ReLU	ReLU
51	'inception_4a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
52	'inception_4a-pool_proj'	Convolution	64 1×1×480 convolutions with stride [1 1] a
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 1×1×512 convolutions with stride [1 1]
56	'inception_4b-relu_1x1'	ReLU	ReLU
57	'inception_4b-3x3_reduce'	Convolution	112 1×1×512 convolutions with stride [1 1]
58	'inception_4b-relu_3x3_reduce'	ReLU	ReLU
59	'inception_4b-3x3'	Convolution	224 3×3×112 convolutions with stride [1 1]
60	'inception_4b-relu_3x3'	ReLU	ReLU
61	'inception_4b-5x5_reduce'	Convolution	24 1×1×512 convolutions with stride [1 1] a
62	'inception_4b-relu_5x5_reduce'	ReLU	ReLU
63	'inception_4b-5x5'	Convolution	64 5×5×24 convolutions with stride [1 1] ar
64	'inception_4b-relu_5x5'	ReLU	ReLU
65	'inception_4b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
66	'inception_4b-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] a
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 1×1×512 convolutions with stride [1 1]
70	'inception_4c-relu_1x1'	ReLU	ReLU
71	'inception_4c-3x3_reduce'	Convolution	128 1×1×512 convolutions with stride [1 1]
72	'inception_4c-relu_3x3_reduce'	ReLU	ReLU
73	'inception_4c-3x3'	Convolution	256 3×3×128 convolutions with stride [1 1]
74	'inception_4c-relu_3x3'	ReLU	ReLU
75	'inception_4c-5x5_reduce'	Convolution	24 1×1×512 convolutions with stride [1 1] a
76	'inception_4c-relu_5x5_reduce'	ReLU	ReLU
77	'inception_4c-5x5'	Convolution	64 5×5×24 convolutions with stride [1 1] ar
78	'inception_4c-relu_5x5'	ReLU	ReLU
79	'inception_4c-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
80	'inception_4c-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] a
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 1×1×512 convolutions with stride [1 1]
84	'inception_4d-relu_1x1'	ReLU	ReLU
85	'inception_4d-3x3_reduce'	Convolution	144 1×1×512 convolutions with stride [1 1]
86	'inception_4d-relu_3x3_reduce'	ReLU	ReLU
87	'inception_4d-3x3'	Convolution	288 3×3×144 convolutions with stride [1 1]
88	'inception_4d-relu_3x3'	ReLU	ReLU
89	'inception_4d-5x5_reduce'	Convolution	32 1×1×512 convolutions with stride [1 1] a
90	'inception_4d-relu_5x5_reduce'	ReLU	ReLU
91	'inception_4d-5x5'	Convolution	64 5×5×32 convolutions with stride [1 1] ar
92	'inception_4d-relu_5x5'	ReLU	ReLU
93	'inception_4d-pool'	Max Pooling	3×3 max pooling with stride [1 1] and paddi
94	'inception_4d-pool_proj'	Convolution	64 1×1×512 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 1×1×528 convolutions with stride [1 1]
98	'inception_4e-relu_1x1'	ReLU	ReLU
99	'inception_4e-3x3_reduce'	Convolution	160 1×1×528 convolutions with stride [1 1]
100	'inception_4e-relu_3x3_reduce'	ReLU	ReLU
101	'inception_4e-3x3'	Convolution	320 3×3×160 convolutions with stride [1 1]
102	'inception_4e-relu_3x3'	ReLU	ReLU
103	'inception_4e-5x5_reduce'	Convolution	32 1×1×528 convolutions with stride [1 1]
104	'inception_4e-relu_5x5_reduce'	ReLU	ReLU
105	'inception_4e-5x5'	Convolution	128 5×5×32 convolutions with stride [1 1]
106	'inception_4e-relu_5x5'	ReLU	ReLU
107	'inception_4e-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padd
108	'inception_4e-pool_proj'	Convolution	128 1×1×528 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	3×3 max pooling with stride [2 2] and padd
112	'inception_5a-1x1'	Convolution	256 1×1×832 convolutions with stride [1 1]
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 padd
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 padd
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

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

```
lgraph =
  LayerGraph with properties:

         Layers: [144×1 nnet.cnn.layer.Layer]
    Connections: [170×2 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: [1518×2 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 = delete(gcp("nocreate"));
end
```

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

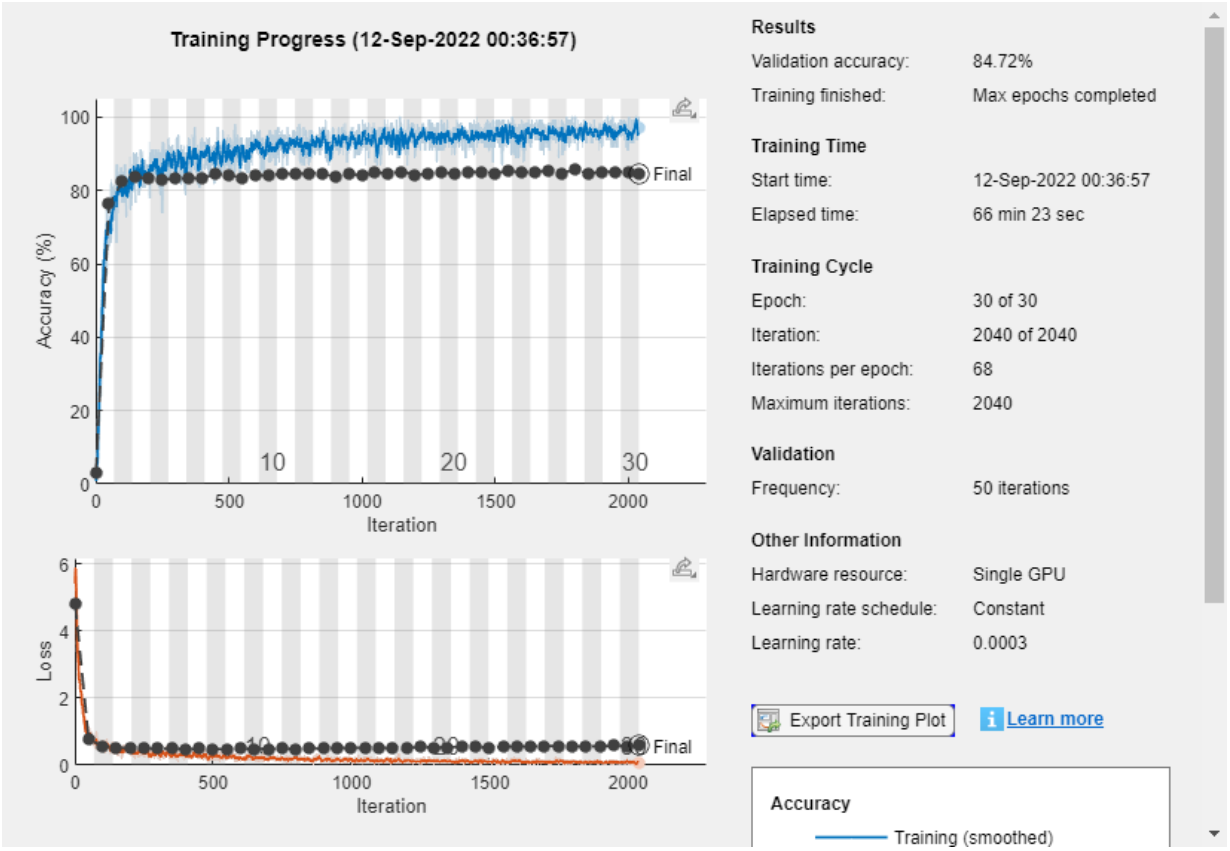
Train the network

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  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learnin
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss     |      Rate
|=========================================================================================================================
|       1 |           1 |       00:00:43 |        2.00% |        2.90% |       5.4207 |       4.8393 |          0.00
|       1 |          50 |       00:03:51 |       68.00% |       76.28% |       0.9903 |       0.7692 |          0.00
|       2 |         100 |       00:05:36 |       77.00% |       82.35% |       0.6545 |       0.5691 |          0.00
|       3 |         150 |       00:08:17 |       77.00% |       83.66% |       0.6284 |       0.5208 |          0.00
|       3 |         200 |       00:10:40 |       86.00% |       83.27% |       0.3795 |       0.5140 |          0.00
|       4 |         250 |       00:12:09 |       74.00% |       83.00% |       0.6043 |       0.5028 |          0.00
|       5 |         300 |       00:13:41 |       82.00% |       83.14% |       0.4168 |       0.4910 |          0.00
|       6 |         350 |       00:15:14 |       90.00% |       83.40% |       0.3156 |       0.4955 |          0.00
|       6 |         400 |       00:16:56 |       92.00% |       83.27% |       0.2477 |       0.4823 |          0.00
|       7 |         450 |       00:18:34 |       88.00% |       84.45% |       0.3097 |       0.4899 |          0.00
|       8 |         500 |       00:20:00 |       89.00% |       84.32% |       0.3145 |       0.4852 |          0.00
|       9 |         550 |       00:21:36 |       87.00% |       83.40% |       0.3420 |       0.4753 |          0.00
|       9 |         600 |       00:23:07 |       94.00% |       83.93% |       0.1320 |       0.4913 |          0.00
|      10 |         650 |       00:24:45 |       93.00% |       84.19% |       0.1712 |       0.4859 |          0.00
|      11 |         700 |       00:26:20 |       91.00% |       84.58% |       0.2574 |       0.4875 |          0.00
|      12 |         750 |       00:27:55 |       94.00% |       84.45% |       0.1949 |       0.4947 |          0.00
```

```
|      12 |         800 |       00:29:40 |       90.00% |       84.72% |       0.2356 |       0.4847 |          0.00
|      13 |         850 |       00:31:11 |       96.00% |       84.72% |       0.1303 |       0.4959 |          0.00
|      14 |         900 |       00:32:37 |       92.00% |       83.66% |       0.1815 |       0.5066 |          0.00
|      14 |         950 |       00:34:04 |       94.00% |       84.58% |       0.1551 |       0.5022 |          0.00
|      15 |        1000 |       00:35:27 |       94.00% |       84.06% |       0.1622 |       0.5221 |          0.00
|      16 |        1050 |       00:36:51 |       94.00% |       84.85% |       0.1071 |       0.5093 |          0.00
|      17 |        1100 |       00:38:16 |       94.00% |       84.72% |       0.1587 |       0.5121 |          0.00
|      17 |        1150 |       00:39:43 |       95.00% |       85.11% |       0.1065 |       0.5129 |          0.00
|      18 |        1200 |       00:41:09 |       91.00% |       84.32% |       0.1998 |       0.5185 |          0.00
|      19 |        1250 |       00:42:33 |       94.00% |       84.45% |       0.1188 |       0.5356 |          0.00
|      20 |        1300 |       00:43:56 |       99.00% |       84.85% |       0.0384 |       0.5280 |          0.00
|      20 |        1350 |       00:45:20 |       97.00% |       84.58% |       0.0765 |       0.5326 |          0.00
|      21 |        1400 |       00:46:45 |       92.00% |       84.98% |       0.1803 |       0.5525 |          0.00
|      22 |        1450 |       00:48:07 |       97.00% |       85.11% |       0.1075 |       0.5343 |          0.00
|      23 |        1500 |       00:49:30 |       97.00% |       84.72% |       0.0963 |       0.5313 |          0.00
|      23 |        1550 |       00:50:56 |       99.00% |       85.24% |       0.0377 |       0.5527 |          0.00
|      24 |        1600 |       00:52:21 |       95.00% |       84.98% |       0.1005 |       0.5495 |          0.00
|      25 |        1650 |       00:53:44 |       98.00% |       84.98% |       0.0745 |       0.5571 |          0.00
|      25 |        1700 |       00:55:09 |       98.00% |       85.51% |       0.0675 |       0.5659 |          0.00
|      26 |        1750 |       00:56:36 |       96.00% |       84.72% |       0.1010 |       0.5765 |          0.00
|      27 |        1800 |       00:58:03 |       94.00% |       85.64% |       0.1419 |       0.5656 |          0.00
|      28 |        1850 |       00:59:29 |       95.00% |       84.58% |       0.0760 |       0.5596 |          0.00
|      28 |        1900 |       01:00:52 |       96.00% |       85.11% |       0.0863 |       0.5674 |          0.00
|      29 |        1950 |       01:02:21 |       96.00% |       85.11% |       0.0707 |       0.5799 |          0.00
|      30 |        2000 |       01:03:55 |       91.00% |       84.98% |       0.1746 |       0.5711 |          0.00
|      30 |        2040 |       01:06:18 |       97.00% |       84.72% |       0.0593 |       0.5849 |          0.00
|=========|=============|================|==============|==============|==============|==============|==============
Training finished: Max epochs completed.
```




```
FlowerNet =
  DAGNetwork with properties:

         Layers: [144×1 nnet.cnn.layer.Layer]
    Connections: [170×2 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 = 1518×39 table
```

...

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
1	Bengal	0.9886	0	0	0	0.0104
2	Bengal	0.9997	0	0	0	0.0003
3	Bengal	0.8092	0	0	0	0.1908
4	Bengal	0.8953	0	0	0	0.1047
5	Bengal	0.9751	0	0	0	0.0203
6	Bengal	0.9977	0	0	0	0.0023
7	Bengal	0.9516	0	0	0	0.0482
8	Egyptian Mau	0.1537	0	0	0	0.8463
9	Bengal	0.9922	0	0.0003	0.0001	0.0033
10	Bengal	0.9979	0	0	0	0.0021
11	Bengal	0.9979	0	0	0	0.0021
12	Bengal	0.9358	0	0	0	0.0641
13	Egyptian Mau	0.2847	0	0	0	0.7084
14	Bengal	0.9643	0	0	0	0.0357
15	Egyptian Mau	0.3923	0	0	0	0.6077
16	Bengal	0.9884	0	0	0	0.0116
17	Egyptian Mau	0.3150	0	0	0.0008	0.6836
18	Bengal	0.6360	0	0	0	0.3407
19	Bengal	0.7476	0	0	0	0.2524
20	Bengal	0.9806	0	0	0	0.0193
21	Birman	0	0.9975	0	0	0
22	Birman	0	0.9944	0	0	0
23	Birman	0	0.9951	0	0	0

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
24	Birman	0	0.8545	0	0	0
25	Birman	0	0.7618	0	0	0
26	Birman	0	1	0	0	0
27	Birman	0	0.9968	0	0	0
28	Birman	0	0.7254	0	0	0
29	Birman	0	0.9999	0	0	0
30	Birman	0	1	0	0	0
31	Birman	0	0.9952	0	0	0
32	Ragdoll	0	0.1349	0	0	0
33	Birman	0	0.9958	0	0	0
34	Ragdoll	0.0002	0.1297	0.0002	0.0002	0.0004
35	Ragdoll	0	0.3294	0	0	0
36	Ragdoll	0	0.0001	0	0	0
37	Ragdoll	0	0.0265	0	0	0
38	Birman	0	0.9959	0	0	0
39	Birman	0	0.9907	0	0	0
40	pomeranian	0	0.0027	0	0	0
41	Bombay	0	0	0.9996	0.0004	0
42	Bombay	0	0	0.9999	0	0
43	Bombay	0	0	1	0	0
44	Bombay	0	0	1	0	0
45	Bombay	0	0	0.9999	0	0
46	Bombay	0	0	0.9961	0	0
47	Bombay	0	0	0.9995	0.0001	0
48	Bombay	0.0001	0	0.9999	0	0
49	Bombay	0.0005	0	0.9511	0	0
50	Bombay	0	0	0.9990	0.0010	0
51	Bombay	0	0	0.9988	0.0002	0
52	Bengal	0.4723	0	0.2775	0.0007	0.0017
53	Bombay	0	0	0.9989	0.0001	0
54	Bombay	0	0	1	0	0
55	Bombay	0	0	1	0	0
56	Bombay	0	0	0.9995	0.0004	0

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
57	Bombay	0	0	0.9999	0	0
58	Bombay	0	0	1	0	0
59	Bombay	0	0	0.9933	0.0015	0
60	Bombay	0	0	1	0	0
61	Russian Blue	0.0035	0	0.0002	0.0048	0.0739
62	Egyptian Mau	0.0029	0	0	0.0872	0.9075
63	British_Shorthair	0.0193	0	0.0001	0.8301	0.0115
64	British_Shorthair	0	0	0	0.9997	0
65	Bombay	0.0009	0	0.9967	0.0002	0
66	British_Shorthair	0	0	0.0569	0.9429	0
67	British_Shorthair	0	0	0.0004	0.9834	0
68	Russian Blue	0.0001	0	0.1311	0.0006	0.0001
69	British_Shorthair	0	0	0	0.9553	0.0001
70	British_Shorthair	0	0	0	0.9991	0
71	Russian Blue	0	0	0	0.3301	0
72	British_Shorthair	0	0	0	1	0
73	British_Shorthair	0	0	0	1	0
74	British_Shorthair	0	0	0.0015	0.9950	0
75	British_Shorthair	0	0	0	0.9999	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	0.9997	0
79	British_Shorthair	0	0	0	1	0
80	British_Shorthair	0	0	0.0023	0.9976	0
81	Egyptian Mau	0.0069	0	0	0	0.9931
82	Egyptian Mau	0.0001	0	0	0	0.9999
83	Egyptian Mau	0.0056	0	0	0	0.9944
84	Bombay	0.0023	0	0.9947	0	0.0004
85	Egyptian Mau	0.0146	0	0	0	0.9854
86	Egyptian Mau	0.0119	0	0	0	0.9881
87	Egyptian Mau	0.0033	0	0	0	0.9955
88	abyssinian	0.1232	0	0.0103	0.0048	0.0666
89	Egyptian Mau	0.0059	0	0	0	0.9941

	predicted	Bengal	Birman	Bombay	British_Shorthair	Egyptian Mau
90	Egyptian Mau	0.0002	0	0	0	0.9998
91	Egyptian Mau	0.3114	0	0	0	0.6762
92	Bengal	0.9434	0	0	0	0.0550
93	Egyptian Mau	0.0657	0	0	0	0.9331
94	Egyptian Mau	0.0003	0	0	0	0.9997
95	Egyptian Mau	0.1223	0	0	0	0.8748
96	Egyptian Mau	0.0001	0	0	0	0.9999
97	Egyptian Mau	0.0001	0	0	0	0.9999
98	Egyptian Mau	0.0008	0	0	0	0.9992
99	Egyptian Mau	0.4791	0	0	0	0.5200
100	Egyptian Mau	0.0079	0	0	0	0.9921

⋮

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

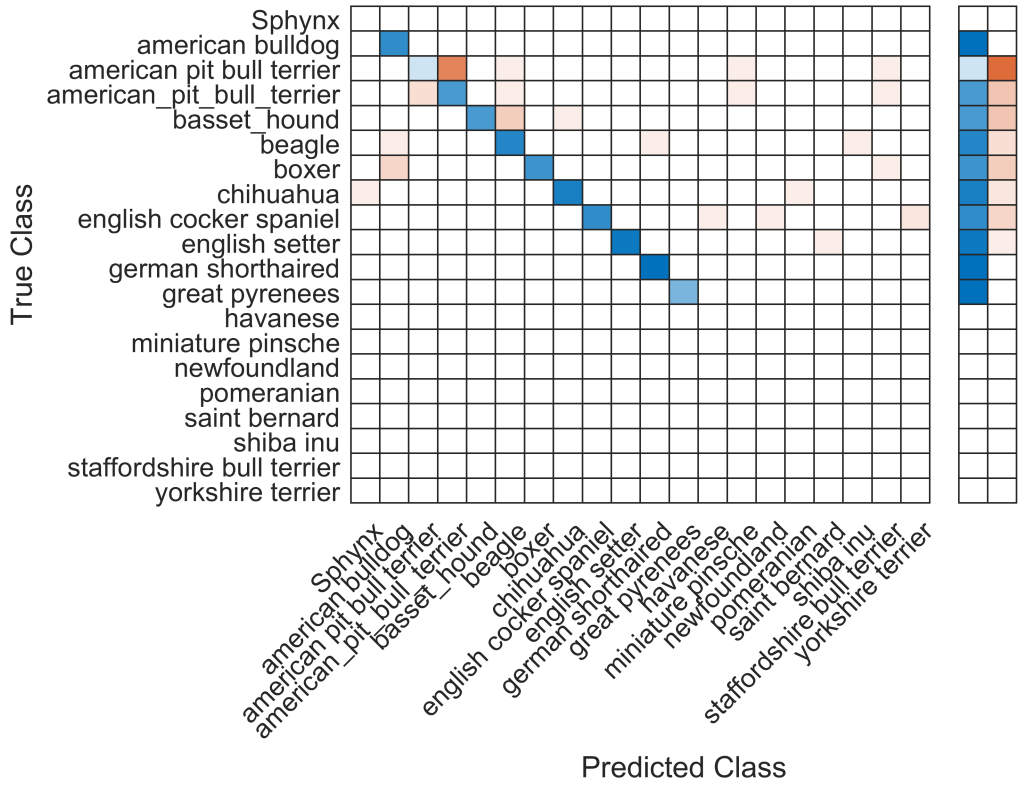
```
ans = 0.1528
```

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","
```

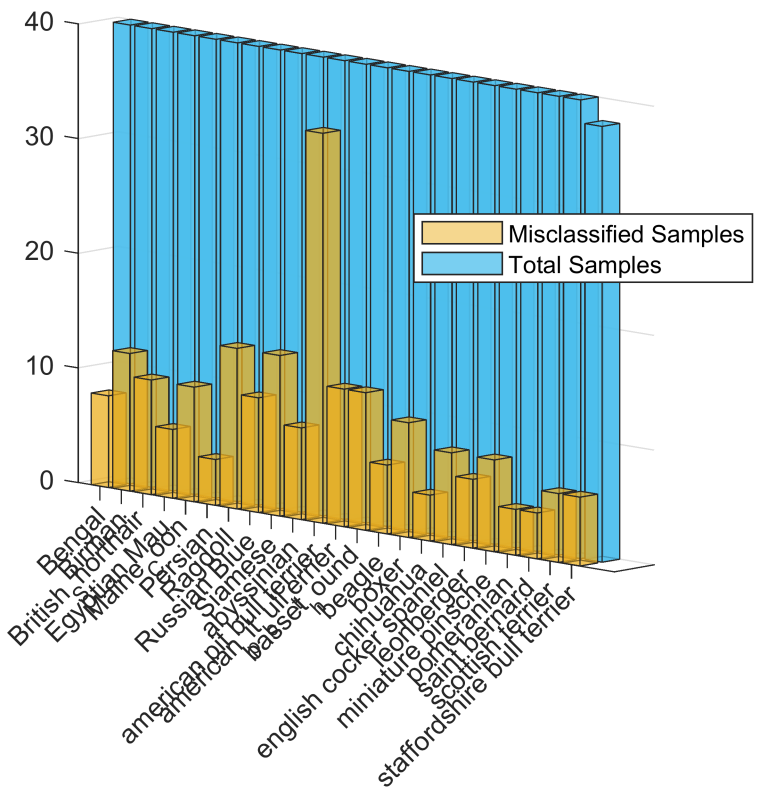



```
cm2 =
  ConfusionMatrixChart with properties:

    NormalizedValues: [20×20 double]
         ClassLabels: [20×1 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 conolutionals 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(acs1)
imshow(imresize(imtile(rescale(acs1)),[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,auds,layer,"MiniBatchSize",1,"OutputAs","rows");
acs1 = activations(FlowerNet,valset(N,:),layer,"OutputAs","rows");
```

```
[idx, d] = knnsearch(acs, acs1, 'K', 5,"NSMethod","kdtree");   %machine learning algorithm of t
%idx(1) = [];
```

```
searchresults = readByIndex(auds,idx);
```

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

```
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(regexp(propName, 'LearnRateFactor$', 'once'))
```

```
            layers(ii).(propName) = 0;
        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),"C
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,searc
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
```