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("FlowerImages/","IncludeSubfolders",true,"LabelSource","folde
```

```
Flowerds =
  ImageDatastore with properties:

                      Files: {
                             ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_06969.jpg';
                             ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_06970.jpg';
                             ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_06971.jpg'
                              ... and 8186 more
                             }
                    Folders: {
                             '/Users/phasuwut/Documents/MATLAB/week_6/lab_3/FlowerImages'
                             }
                     Labels: [alpine sea holly; alpine sea holly; alpine sea holly ... and 8186 more cate
    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: {
                             ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_06969.jpg';
                             ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_06970.jpg';
                             ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_06971.jpg'
                              ... and 7370 more
                             }
                    Folders: {
                             '/Users/phasuwut/Documents/MATLAB/week_6/lab_3/FlowerImages'
                             }
                     Labels: [alpine sea holly; alpine sea holly; alpine sea holly ... and 7370 more cate
    AlternateFileSystemRoots: {}
                   ReadSize: 1
      SupportedOutputFormats: ["png"    "jpg"    "jpeg"    "tif"    "tiff"]
```

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

                       Files: {
                              ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_08085.jpg';
                              ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_08086.jpg';
                              ' .../MATLAB/week_6/lab_3/FlowerImages/alpine sea holly/image_08087.jpg'
                               ... and 813 more
                              }
                     Folders: {
                              '/Users/phasuwut/Documents/MATLAB/week_6/lab_3/FlowerImages'
                              }
                      Labels: [alpine sea holly; alpine sea holly; alpine sea holly ... and 813 more categ
    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,'DataAugmentation', ...
 imageDataAugmenter("RandRotation",[-180 180],"RandXReflection",true))
```

```
auds =
  augmentedImageDatastore with properties:

            NumObservations: 7373
                      Files: {7373×1 cell}
    AlternateFileSystemRoots: {}
               MiniBatchSize: 128
          DataAugmentation: [1×1 imageDataAugmenter]
        ColorPreprocessing: 'none'
                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 = 102×2 table
```

	num	Var2
1	alpine sea holly	39
2	anthurium	95
3	artichoke	70
4	azalea	86
5	ball moss	41
6	balloon flower	44
7	barbeton daisy	114
8	bearded iris	49
9	bee balm	59
10	bird of paradise	77
11	bishop of lland...	98
12	black-eyed susan	49
13	blackberry lily	43
14	blanket flower	44
15	bolero deep blue	36
16	bougainvillea	115
17	bromelia	57
18	buttercup	64
19	californian poppy	92
20	camellia	82
21	canna lily	74
22	canterbury bells	36
23	cape flower	97
24	carnation	47
25	cautleya spicata	45
26	clematis	101
27	colt's foot	78
28	columbine	77
29	common dandelion	83
30	corn poppy	37
31	cyclamen	139
32	daffodil	53
33	desert-rose	57

	num	Var2
34	english marigold	59
35	fire lily	36
36	foxglove	146
37	frangipani	149
38	fritillary	82
39	garden phlox	41
40	gaura	60
41	gazania	70
42	geranium	103
43	giant white aru...	50
44	globe thistle	41
45	globe-flower	37
46	grape hyacinth	37
47	great masterwort	50
48	hard-leaved poc...	54
49	hibiscus	118
50	hippeastrum	68
51	japanese anemone	50
52	king protea	44
53	lenten rose	60
54	lotus	123
55	love in the mist	41
56	magnolia	57
57	mallow	59
58	marigold	60
59	mexican aster	36
60	mexican petunia	74
61	monkshood	41
62	moon orchid	36
63	morning glory	96
64	orange dahlia	60
65	osteospermum	55
66	oxeye daisy	44

	num	Var2
67	passion flower	226
68	pelargonium	64
69	peruvian lily	74
70	petunia	232
71	pincushion flower	53
72	pink primrose	36
73	pink-yellow dah...	98
74	poinsettia	84
75	primula	84
76	prince of wales...	36
77	purple coneflower	77
78	red ginger	38
79	rose	154
80	ruby-lipped cat...	68
81	siam tulip	37
82	silverbush	47
83	snapdragon	78
84	spear thistle	43
85	spring crocus	38
86	stemless gentian	59
87	sunflower	55
88	sweet pea	50
89	sweet william	77
90	sword lily	117
91	thorn apple	108
92	tiger lily	41
93	toad lily	37
94	tree mallow	52
95	tree poppy	56
96	trumpet creeper	52
97	wallflower	176
98	water lily	175
99	watercress	166

	num	Var2
100	wild pansy	77
	⋮	

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

```
Minimum number of training images in a label: 36
```

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,'DataAugmentation', ...
 imageDataAugmenter("RandRotation",[-180 180])))]
```

```
valset = 1632×2 table
```

	input	response
1	224×224×3 uint8	alpine sea holly
2	224×224×3 uint8	alpine sea holly
3	224×224×3 uint8	alpine sea holly
4	224×224×3 uint8	alpine sea holly
5	224×224×3 uint8	anthurium
6	224×224×3 uint8	anthurium
7	224×224×3 uint8	anthurium
8	224×224×3 uint8	anthurium
9	224×224×3 uint8	anthurium
10	224×224×3 uint8	anthurium
11	224×224×3 uint8	anthurium
12	224×224×3 uint8	anthurium
13	224×224×3 uint8	anthurium
14	224×224×3 uint8	anthurium
15	224×224×3 uint8	artichoke
16	224×224×3 uint8	artichoke
17	224×224×3 uint8	artichoke
18	224×224×3 uint8	artichoke
19	224×224×3 uint8	artichoke

	input	response
20	224×224×3 uint8	artichoke
21	224×224×3 uint8	artichoke
22	224×224×3 uint8	artichoke
23	224×224×3 uint8	azalea
24	224×224×3 uint8	azalea
25	224×224×3 uint8	azalea
26	224×224×3 uint8	azalea
27	224×224×3 uint8	azalea
28	224×224×3 uint8	azalea
29	224×224×3 uint8	azalea
30	224×224×3 uint8	azalea
31	224×224×3 uint8	azalea
32	224×224×3 uint8	azalea
33	224×224×3 uint8	ball moss
34	224×224×3 uint8	ball moss
35	224×224×3 uint8	ball moss
36	224×224×3 uint8	ball moss
37	224×224×3 uint8	ball moss
38	224×224×3 uint8	balloon flower
39	224×224×3 uint8	balloon flower
40	224×224×3 uint8	balloon flower
41	224×224×3 uint8	balloon flower
42	224×224×3 uint8	balloon flower
43	224×224×3 uint8	barbeton daisy
44	224×224×3 uint8	barbeton daisy
45	224×224×3 uint8	barbeton daisy
46	224×224×3 uint8	barbeton daisy
47	224×224×3 uint8	barbeton daisy
48	224×224×3 uint8	barbeton daisy
49	224×224×3 uint8	barbeton daisy
50	224×224×3 uint8	barbeton daisy
51	224×224×3 uint8	barbeton daisy
52	224×224×3 uint8	barbeton daisy

	input	response
53	224×224×3 uint8	barbeton daisy
54	224×224×3 uint8	barbeton daisy
55	224×224×3 uint8	barbeton daisy
56	224×224×3 uint8	bearded iris
57	224×224×3 uint8	bearded iris
58	224×224×3 uint8	bearded iris
59	224×224×3 uint8	bearded iris
60	224×224×3 uint8	bearded iris
61	224×224×3 uint8	bee balm
62	224×224×3 uint8	bee balm
63	224×224×3 uint8	bee balm
64	224×224×3 uint8	bee balm
65	224×224×3 uint8	bee balm
66	224×224×3 uint8	bee balm
67	224×224×3 uint8	bee balm
68	224×224×3 uint8	bird of paradise
69	224×224×3 uint8	bird of paradise
70	224×224×3 uint8	bird of paradise
71	224×224×3 uint8	bird of paradise
72	224×224×3 uint8	bird of paradise
73	224×224×3 uint8	bird of paradise
74	224×224×3 uint8	bird of paradise
75	224×224×3 uint8	bird of paradise
76	224×224×3 uint8	bishop of lland...
77	224×224×3 uint8	bishop of lland...
78	224×224×3 uint8	bishop of lland...
79	224×224×3 uint8	bishop of lland...
80	224×224×3 uint8	bishop of lland...
81	224×224×3 uint8	bishop of lland...
82	224×224×3 uint8	bishop of lland...
83	224×224×3 uint8	bishop of lland...
84	224×224×3 uint8	bishop of lland...
85	224×224×3 uint8	bishop of lland...

	input	response
86	224×224×3 uint8	bishop of lland...
87	224×224×3 uint8	black-eyed susan
88	224×224×3 uint8	black-eyed susan
89	224×224×3 uint8	black-eyed susan
90	224×224×3 uint8	black-eyed susan
91	224×224×3 uint8	black-eyed susan
92	224×224×3 uint8	blackberry lily
93	224×224×3 uint8	blackberry lily
94	224×224×3 uint8	blackberry lily
95	224×224×3 uint8	blackberry lily
96	224×224×3 uint8	blackberry lily
97	224×224×3 uint8	blanket flower
98	224×224×3 uint8	blanket flower
99	224×224×3 uint8	blanket flower
100	224×224×3 uint8	blanket flower

⋮

Number of Samples in Validation Set

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

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

a_table = 102×2 table

	num	Var2
1	alpine sea holly	8
2	anthurium	20
3	artichoke	16
4	azalea	20
5	ball moss	10
6	balloon flower	10
7	barbeton daisy	26
8	bearded iris	10
9	bee balm	14
10	bird of paradise	16
11	bishop of lland...	22

	num	Var2
12	black-eyed susan	10
13	blackberry lily	10
14	blanket flower	10
15	bolero deep blue	8
16	bougainvillea	26
17	bromelia	12
18	buttercup	14
19	californian poppy	20
20	camellia	18
21	canna lily	16
22	canterbury bells	8
23	cape flower	22
24	carnation	10
25	cautleya spicata	10
26	clematis	22
27	colt's foot	18
28	columbine	18
29	common dandelion	18
30	corn poppy	8
31	cyclamen	30
32	daffodil	12
33	desert-rose	12
34	english marigold	12
35	fire lily	8
36	foxglove	32
37	frangipani	34
38	fritillary	18
39	garden phlox	8
40	gaura	14
41	gazania	16
42	geranium	22
43	giant white aru...	12
44	globe thistle	8

	num	Var2
45	globe-flower	8
46	grape hyacinth	8
47	great masterwort	12
48	hard-leaved poc...	12
49	hibiscus	26
50	hippeastrum	16
51	japanese anemone	10
52	king protea	10
53	lenten rose	14
54	lotus	28
55	love in the mist	10
56	magnolia	12
57	mallow	14
58	marigold	14
59	mexican aster	8
60	mexican petunia	16
61	monkshood	10
62	moon orchid	8
63	morning glory	22
64	orange dahlia	14
65	osteospermum	12
66	oxeye daisy	10
67	passion flower	50
68	pelargonium	14
69	peruvian lily	16
70	petunia	52
71	pincushion flower	12
72	pink primrose	8
73	pink-yellow dah...	22
74	poinsettia	18
75	primula	18
76	prince of wales...	8
77	purple coneflower	16

	num	Var2
78	red ginger	8
79	rose	34
80	ruby-lipped cat...	14
81	siam tulip	8
82	silverbush	10
83	snapdragon	18
84	spear thistle	10
85	spring crocus	8
86	stemless gentian	14
87	sunflower	12
88	sweet pea	12
89	sweet william	16
90	sword lily	26
91	thorn apple	24
92	tiger lily	8
93	toad lily	8
94	tree mallow	12
95	tree poppy	12
96	trumpet creeper	12
97	wallflower	40
98	water lily	38
99	watercress	36
100	wild pansy	16

⋮

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

```
ans = 8
```

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(102,"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
     2   'conv1-7x7_s2'                  Convolution                 64 7×7×3 convolutions with stride
     3   'conv1-relu_7x7'                ReLU                        ReLU
     4   'pool1-3x3_s2'                  Max Pooling                 3×3 max pooling with stride [2  2]
     5   'pool1-norm1'                   Cross Channel Normalization   cross channel normalization with 5
     6   'conv2-3x3_reduce'              Convolution                 64 1×1×64 convolutions with stride
     7   'conv2-relu_3x3_reduce'         ReLU                        ReLU
     8   'conv2-3x3'                     Convolution                 192 3×3×64 convolutions with strid
     9   'conv2-relu_3x3'                ReLU                        ReLU
    10   'conv2-norm2'                   Cross Channel Normalization   cross channel normalization with 5
    11   'pool2-3x3_s2'                  Max Pooling                 3×3 max pooling with stride [2  2]
    12   'inception_3a-1x1'              Convolution                 64 1×1×192 convolutions with strid
    13   'inception_3a-relu_1x1'         ReLU                        ReLU
    14   'inception_3a-3x3_reduce'       Convolution                 96 1×1×192 convolutions with strid
    15   'inception_3a-relu_3x3_reduce'  ReLU                        ReLU
    16   'inception_3a-3x3'              Convolution                 128 3×3×96 convolutions with strid
    17   'inception_3a-relu_3x3'         ReLU                        ReLU
    18   'inception_3a-5x5_reduce'       Convolution                 16 1×1×192 convolutions with strid
    19   'inception_3a-relu_5x5_reduce'  ReLU                        ReLU
    20   'inception_3a-5x5'              Convolution                 32 5×5×16 convolutions with stride
    21   'inception_3a-relu_5x5'         ReLU                        ReLU
    22   'inception_3a-pool'             Max Pooling                 3×3 max pooling with stride [1  1]
    23   'inception_3a-pool_proj'        Convolution                 32 1×1×192 convolutions with strid
    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 stri
    27   'inception_3b-relu_1x1'         ReLU                        ReLU
    28   'inception_3b-3x3_reduce'       Convolution                 128 1×1×256 convolutions with stri
    29   'inception_3b-relu_3x3_reduce'  ReLU                        ReLU
    30   'inception_3b-3x3'              Convolution                 192 3×3×128 convolutions with stri
    31   'inception_3b-relu_3x3'         ReLU                        ReLU
    32   'inception_3b-5x5_reduce'       Convolution                 32 1×1×256 convolutions with strid
    33   'inception_3b-relu_5x5_reduce'  ReLU                        ReLU
    34   'inception_3b-5x5'              Convolution                 96 5×5×32 convolutions with stride
    35   'inception_3b-relu_5x5'         ReLU                        ReLU
    36   'inception_3b-pool'             Max Pooling                 3×3 max pooling with stride [1  1]
    37   'inception_3b-pool_proj'        Convolution                 64 1×1×256 convolutions with strid
    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]
    41   'inception_4a-1x1'              Convolution                 192 1×1×480 convolutions with stri
    42   'inception_4a-relu_1x1'         ReLU                        ReLU
    43   'inception_4a-3x3_reduce'       Convolution                 96 1×1×480 convolutions with strid
    44   'inception_4a-relu_3x3_reduce'  ReLU                        ReLU
    45   'inception_4a-3x3'              Convolution                 208 3×3×96 convolutions with strid
    46   'inception_4a-relu_3x3'         ReLU                        ReLU
    47   'inception_4a-5x5_reduce'       Convolution                 16 1×1×480 convolutions with strid
    48   'inception_4a-relu_5x5_reduce'  ReLU                        ReLU
    49   'inception_4a-5x5'              Convolution                 48 5×5×16 convolutions with stride
    50   'inception_4a-relu_5x5'         ReLU                        ReLU
    51   'inception_4a-pool'             Max Pooling                 3×3 max pooling with stride [1  1]
    52   'inception_4a-pool_proj'        Convolution                 64 1×1×480 convolutions with strid
    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 stri
56   'inception_4b-relu_1x1'          ReLU                       ReLU
57   'inception_4b-3x3_reduce'        Convolution                112 1×1×512 convolutions with stri
58   'inception_4b-relu_3x3_reduce'   ReLU                       ReLU
59   'inception_4b-3x3'               Convolution                224 3×3×112 convolutions with stri
60   'inception_4b-relu_3x3'          ReLU                       ReLU
61   'inception_4b-5x5_reduce'        Convolution                24 1×1×512 convolutions with strid
62   'inception_4b-relu_5x5_reduce'   ReLU                       ReLU
63   'inception_4b-5x5'               Convolution                64 5×5×24 convolutions with stride
64   'inception_4b-relu_5x5'          ReLU                       ReLU
65   'inception_4b-pool'              Max Pooling                3×3 max pooling with stride [1  1]
66   'inception_4b-pool_proj'         Convolution                64 1×1×512 convolutions with strid
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 stri
70   'inception_4c-relu_1x1'          ReLU                       ReLU
71   'inception_4c-3x3_reduce'        Convolution                128 1×1×512 convolutions with stri
72   'inception_4c-relu_3x3_reduce'   ReLU                       ReLU
73   'inception_4c-3x3'               Convolution                256 3×3×128 convolutions with stri
74   'inception_4c-relu_3x3'          ReLU                       ReLU
75   'inception_4c-5x5_reduce'        Convolution                24 1×1×512 convolutions with strid
76   'inception_4c-relu_5x5_reduce'   ReLU                       ReLU
77   'inception_4c-5x5'               Convolution                64 5×5×24 convolutions with stride
78   'inception_4c-relu_5x5'          ReLU                       ReLU
79   'inception_4c-pool'              Max Pooling                3×3 max pooling with stride [1  1]
80   'inception_4c-pool_proj'         Convolution                64 1×1×512 convolutions with strid
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 stri
84   'inception_4d-relu_1x1'          ReLU                       ReLU
85   'inception_4d-3x3_reduce'        Convolution                144 1×1×512 convolutions with stri
86   'inception_4d-relu_3x3_reduce'   ReLU                       ReLU
87   'inception_4d-3x3'               Convolution                288 3×3×144 convolutions with stri
88   'inception_4d-relu_3x3'          ReLU                       ReLU
89   'inception_4d-5x5_reduce'        Convolution                32 1×1×512 convolutions with strid
90   'inception_4d-relu_5x5_reduce'   ReLU                       ReLU
91   'inception_4d-5x5'               Convolution                64 5×5×32 convolutions with stride
92   'inception_4d-relu_5x5'          ReLU                       ReLU
93   'inception_4d-pool'              Max Pooling                3×3 max pooling with stride [1  1]
94   'inception_4d-pool_proj'         Convolution                64 1×1×512 convolutions with strid
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 stri
98   'inception_4e-relu_1x1'          ReLU                       ReLU
99   'inception_4e-3x3_reduce'        Convolution                160 1×1×528 convolutions with stri
100   'inception_4e-relu_3x3_reduce'   ReLU                       ReLU
101   'inception_4e-3x3'               Convolution                320 3×3×160 convolutions with str
102   'inception_4e-relu_3x3'          ReLU                       ReLU
103   'inception_4e-5x5_reduce'        Convolution                32 1×1×528 convolutions with stri
104   'inception_4e-relu_5x5_reduce'   ReLU                       ReLU
105   'inception_4e-5x5'               Convolution                128 5×5×32 convolutions with stri
106   'inception_4e-relu_5x5'          ReLU                       ReLU
107   'inception_4e-pool'              Max Pooling                3×3 max pooling with stride [1  1
108   'inception_4e-pool_proj'         Convolution                128 1×1×528 convolutions with str
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
112   'inception_5a-1x1'               Convolution                256 1×1×832 convolutions with str
113   'inception_5a-relu_1x1'          ReLU                       ReLU
114   'inception_5a-3x3_reduce'        Convolution                160 1×1×832 convolutions with str
115   'inception_5a-relu_3x3_reduce'   ReLU                       ReLU
116   'inception_5a-3x3'               Convolution                320 3×3×160 convolutions with str
117   'inception_5a-relu_3x3'          ReLU                       ReLU
118   'inception_5a-5x5_reduce'        Convolution                32 1×1×832 convolutions with stri
```

```
119   'inception_5a-relu_5x5_reduce'   ReLU                          ReLU
120   'inception_5a-5x5'               Convolution                   128 5×5×32 convolutions with stri
121   'inception_5a-relu_5x5'          ReLU                          ReLU
122   'inception_5a-pool'              Max Pooling                   3×3 max pooling with stride [1  1
123   'inception_5a-pool_proj'         Convolution                   128 1×1×832 convolutions with str
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 str
127   'inception_5b-relu_1x1'          ReLU                          ReLU
128   'inception_5b-3x3_reduce'        Convolution                   192 1×1×832 convolutions with str
129   'inception_5b-relu_3x3_reduce'   ReLU                          ReLU
130   'inception_5b-3x3'               Convolution                   384 3×3×192 convolutions with str
131   'inception_5b-relu_3x3'          ReLU                          ReLU
132   'inception_5b-5x5_reduce'        Convolution                   48 1×1×832 convolutions with stri
133   'inception_5b-relu_5x5_reduce'   ReLU                          ReLU
134   'inception_5b-5x5'               Convolution                   128 5×5×48 convolutions with stri
135   'inception_5b-relu_5x5'          ReLU                          ReLU
136   'inception_5b-pool'              Max Pooling                   3×3 max pooling with stride [1  1
137   'inception_5b-pool_proj'         Convolution                   128 1×1×832 convolutions with str
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               102 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'
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_...

	Source	Destination
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'
48	'pool3-3x3_s2'	'inception_4a-5x5_reduce'
49	'pool3-3x3_s2'	'inception_4a-pool'
50	'inception_4a-1x1'	'inception_4a-relu_1x1'

	Source	Destination
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'
81	'inception_4b-output'	'inception_4c-3x3_reduce'
82	'inception_4b-output'	'inception_4c-5x5_reduce'
83	'inception_4b-output'	'inception_4c-pool'

	Source	Destination
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
     2   'conv1-7x7_s2'                  Convolution                 64 7×7×3 convolutions with stride
     3   'conv1-relu_7x7'                ReLU                        ReLU
     4   'pool1-3x3_s2'                  Max Pooling                 3×3 max pooling with stride [2  2]
     5   'pool1-norm1'                   Cross Channel Normalization cross channel normalization with 5
     6   'conv2-3x3_reduce'              Convolution                 64 1×1×64 convolutions with stride
     7   'conv2-relu_3x3_reduce'         ReLU                        ReLU
     8   'conv2-3x3'                     Convolution                 192 3×3×64 convolutions with strid
     9   'conv2-relu_3x3'                ReLU                        ReLU
    10   'conv2-norm2'                   Cross Channel Normalization cross channel normalization with 5
    11   'pool2-3x3_s2'                  Max Pooling                 3×3 max pooling with stride [2  2]
    12   'inception_3a-1x1'              Convolution                 64 1×1×192 convolutions with strid
    13   'inception_3a-relu_1x1'         ReLU                        ReLU
    14   'inception_3a-3x3_reduce'       Convolution                 96 1×1×192 convolutions with strid
    15   'inception_3a-relu_3x3_reduce'  ReLU                        ReLU
    16   'inception_3a-3x3'              Convolution                 128 3×3×96 convolutions with strid
    17   'inception_3a-relu_3x3'         ReLU                        ReLU
    18   'inception_3a-5x5_reduce'       Convolution                 16 1×1×192 convolutions with strid
    19   'inception_3a-relu_5x5_reduce'  ReLU                        ReLU
    20   'inception_3a-5x5'              Convolution                 32 5×5×16 convolutions with stride
    21   'inception_3a-relu_5x5'         ReLU                        ReLU
    22   'inception_3a-pool'             Max Pooling                 3×3 max pooling with stride [1  1]
```

```
23   'inception_3a-pool_proj'         Convolution                32 1×1×192 convolutions with stric
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 stri
27   'inception_3b-relu_1x1'          ReLU                       ReLU
28   'inception_3b-3x3_reduce'        Convolution                128 1×1×256 convolutions with stri
29   'inception_3b-relu_3x3_reduce'   ReLU                       ReLU
30   'inception_3b-3x3'               Convolution                192 3×3×128 convolutions with stri
31   'inception_3b-relu_3x3'          ReLU                       ReLU
32   'inception_3b-5x5_reduce'        Convolution                32 1×1×256 convolutions with stric
33   'inception_3b-relu_5x5_reduce'   ReLU                       ReLU
34   'inception_3b-5x5'               Convolution                96 5×5×32 convolutions with stride
35   'inception_3b-relu_5x5'          ReLU                       ReLU
36   'inception_3b-pool'              Max Pooling                3×3 max pooling with stride [1  1]
37   'inception_3b-pool_proj'         Convolution                64 1×1×256 convolutions with stric
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]
41   'inception_4a-1x1'               Convolution                192 1×1×480 convolutions with stri
42   'inception_4a-relu_1x1'          ReLU                       ReLU
43   'inception_4a-3x3_reduce'        Convolution                96 1×1×480 convolutions with stric
44   'inception_4a-relu_3x3_reduce'   ReLU                       ReLU
45   'inception_4a-3x3'               Convolution                208 3×3×96 convolutions with stric
46   'inception_4a-relu_3x3'          ReLU                       ReLU
47   'inception_4a-5x5_reduce'        Convolution                16 1×1×480 convolutions with stric
48   'inception_4a-relu_5x5_reduce'   ReLU                       ReLU
49   'inception_4a-5x5'               Convolution                48 5×5×16 convolutions with stride
50   'inception_4a-relu_5x5'          ReLU                       ReLU
51   'inception_4a-pool'              Max Pooling                3×3 max pooling with stride [1  1]
52   'inception_4a-pool_proj'         Convolution                64 1×1×480 convolutions with stric
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 stri
56   'inception_4b-relu_1x1'          ReLU                       ReLU
57   'inception_4b-3x3_reduce'        Convolution                112 1×1×512 convolutions with stri
58   'inception_4b-relu_3x3_reduce'   ReLU                       ReLU
59   'inception_4b-3x3'               Convolution                224 3×3×112 convolutions with stri
60   'inception_4b-relu_3x3'          ReLU                       ReLU
61   'inception_4b-5x5_reduce'        Convolution                24 1×1×512 convolutions with stric
62   'inception_4b-relu_5x5_reduce'   ReLU                       ReLU
63   'inception_4b-5x5'               Convolution                64 5×5×24 convolutions with stride
64   'inception_4b-relu_5x5'          ReLU                       ReLU
65   'inception_4b-pool'              Max Pooling                3×3 max pooling with stride [1  1]
66   'inception_4b-pool_proj'         Convolution                64 1×1×512 convolutions with stric
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 stri
70   'inception_4c-relu_1x1'          ReLU                       ReLU
71   'inception_4c-3x3_reduce'        Convolution                128 1×1×512 convolutions with stri
72   'inception_4c-relu_3x3_reduce'   ReLU                       ReLU
73   'inception_4c-3x3'               Convolution                256 3×3×128 convolutions with stri
74   'inception_4c-relu_3x3'          ReLU                       ReLU
75   'inception_4c-5x5_reduce'        Convolution                24 1×1×512 convolutions with stric
76   'inception_4c-relu_5x5_reduce'   ReLU                       ReLU
77   'inception_4c-5x5'               Convolution                64 5×5×24 convolutions with stride
78   'inception_4c-relu_5x5'          ReLU                       ReLU
79   'inception_4c-pool'              Max Pooling                3×3 max pooling with stride [1  1]
80   'inception_4c-pool_proj'         Convolution                64 1×1×512 convolutions with stric
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 stri
84   'inception_4d-relu_1x1'          ReLU                       ReLU
85   'inception_4d-3x3_reduce'        Convolution                144 1×1×512 convolutions with stri
86   'inception_4d-relu_3x3_reduce'   ReLU                       ReLU
```

87	'inception_4d-3x3'	Convolution	288 3×3×144 convolutions with stri
88	'inception_4d-relu_3x3'	ReLU	ReLU
89	'inception_4d-5x5_reduce'	Convolution	32 1×1×512 convolutions with strid
90	'inception_4d-relu_5x5_reduce'	ReLU	ReLU
91	'inception_4d-5x5'	Convolution	64 5×5×32 convolutions with stride
92	'inception_4d-relu_5x5'	ReLU	ReLU
93	'inception_4d-pool'	Max Pooling	3×3 max pooling with stride [1 1]
94	'inception_4d-pool_proj'	Convolution	64 1×1×512 convolutions with strid
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 stri
98	'inception_4e-relu_1x1'	ReLU	ReLU
99	'inception_4e-3x3_reduce'	Convolution	160 1×1×528 convolutions with stri
100	'inception_4e-relu_3x3_reduce'	ReLU	ReLU
101	'inception_4e-3x3'	Convolution	320 3×3×160 convolutions with str
102	'inception_4e-relu_3x3'	ReLU	ReLU
103	'inception_4e-5x5_reduce'	Convolution	32 1×1×528 convolutions with stri
104	'inception_4e-relu_5x5_reduce'	ReLU	ReLU
105	'inception_4e-5x5'	Convolution	128 5×5×32 convolutions with stri
106	'inception_4e-relu_5x5'	ReLU	ReLU
107	'inception_4e-pool'	Max Pooling	3×3 max pooling with stride [1 1
108	'inception_4e-pool_proj'	Convolution	128 1×1×528 convolutions with str
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
112	'inception_5a-1x1'	Convolution	256 1×1×832 convolutions with str
113	'inception_5a-relu_1x1'	ReLU	ReLU
114	'inception_5a-3x3_reduce'	Convolution	160 1×1×832 convolutions with str
115	'inception_5a-relu_3x3_reduce'	ReLU	ReLU
116	'inception_5a-3x3'	Convolution	320 3×3×160 convolutions with str
117	'inception_5a-relu_3x3'	ReLU	ReLU
118	'inception_5a-5x5_reduce'	Convolution	32 1×1×832 convolutions with stri
119	'inception_5a-relu_5x5_reduce'	ReLU	ReLU
120	'inception_5a-5x5'	Convolution	128 5×5×32 convolutions with stri
121	'inception_5a-relu_5x5'	ReLU	ReLU
122	'inception_5a-pool'	Max Pooling	3×3 max pooling with stride [1 1
123	'inception_5a-pool_proj'	Convolution	128 1×1×832 convolutions with str
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 str
127	'inception_5b-relu_1x1'	ReLU	ReLU
128	'inception_5b-3x3_reduce'	Convolution	192 1×1×832 convolutions with str
129	'inception_5b-relu_3x3_reduce'	ReLU	ReLU
130	'inception_5b-3x3'	Convolution	384 3×3×192 convolutions with str
131	'inception_5b-relu_3x3'	ReLU	ReLU
132	'inception_5b-5x5_reduce'	Convolution	48 1×1×832 convolutions with stri
133	'inception_5b-relu_5x5_reduce'	ReLU	ReLU
134	'inception_5b-5x5'	Convolution	128 5×5×48 convolutions with stri
135	'inception_5b-relu_5x5'	ReLU	ReLU
136	'inception_5b-pool'	Max Pooling	3×3 max pooling with stride [1 1
137	'inception_5b-pool_proj'	Convolution	128 1×1×832 convolutions with str
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	102 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',3, ...
    '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: 3
                   MiniBatchSize: 100
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: [1632×2 table]
              ValidationFrequency: 50
               ValidationPatience: Inf
                         Shuffle: 'every-epoch'
                  CheckpointPath: ''
              CheckpointFrequency: 1
         CheckpointFrequencyUnit: 'epoch'
```

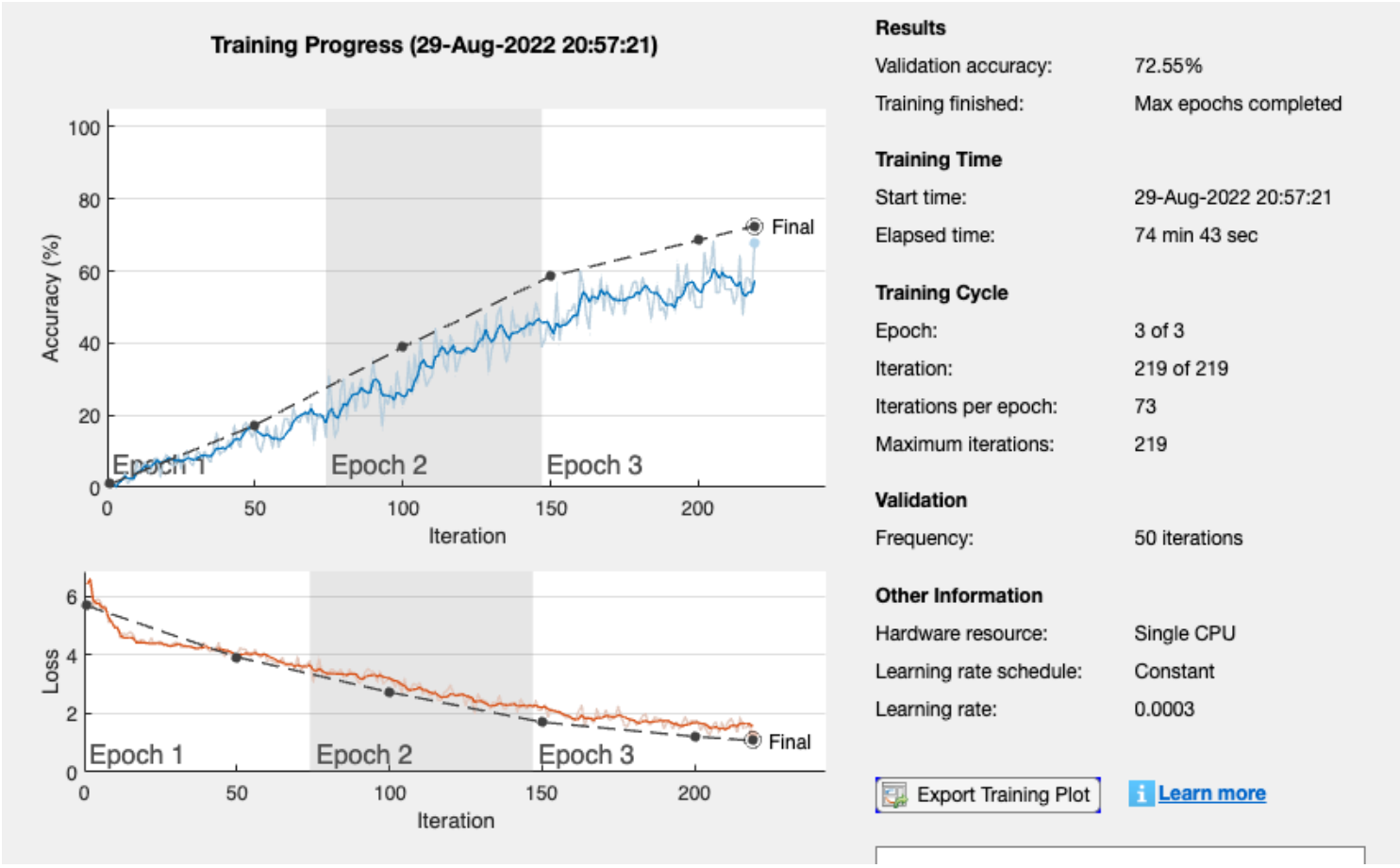
```
          ExecutionEnvironment: 'auto'
                    WorkerLoad: []
                     OutputFcn: []
                         Plots: 'training-progress'
                SequenceLength: 'longest'
            SequencePaddingValue: 0
        SequencePaddingDirection: 'right'
            DispatchInBackground: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

Train the network

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

```
FlowerNet = trainNetwork(auds,lgraph_1,options)
```

```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================================================
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Ba
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss     |
|=========================================================================================================================
|       1 |           1 |       00:02:12 |        1.00% |        0.92% |       6.4247 |       5.6927 |
|       1 |          50 |       00:21:05 |       16.00% |       17.22% |       3.8981 |       3.9357 |
|       2 |         100 |       00:37:36 |       23.00% |       38.91% |       3.2305 |       2.7351 |
|       3 |         150 |       00:52:40 |       42.00% |       58.52% |       2.1510 |       1.7075 |
|       3 |         200 |       01:06:39 |       50.00% |       68.57% |       1.7558 |       1.2158 |
|       3 |         219 |       01:12:56 |       68.00% |       72.55% |       1.2927 |       1.0775 |
|=========================================================================================================================
Training finished: Max epochs completed.
```




```
FlowerNet =
  DAGNetwork with properties:

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

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
```

```
tableofscores = 1632×103 table
```

...

	predicted	alpine sea holly	anthurium	artichoke	azalea	ball moss
1	alpine sea holly	0.9084	0.0001	0.0008	0.0002	0.0006
2	alpine sea holly	0.8348	0	0.0035	0.0002	0.0010
3	alpine sea holly	0.8467	0.0001	0.0011	0.0010	0.0012

	predicted	alpine sea holly	anthurium	artichoke	azalea	ball moss
4	alpine sea holly	0.5722	0.0001	0.0077	0.0054	0.0044
5	anthurium	0	0.9475	0	0	0.0001
6	anthurium	0	0.7973	0	0.0011	0.0004
7	anthurium	0	0.6346	0	0.0007	0.0007
8	anthurium	0.0003	0.5464	0.0016	0.0044	0.0014
9	anthurium	0	0.8393	0	0.0002	0.0004
10	morning glory	0.0005	0.0588	0.0001	0.0022	0.0016
11	anthurium	0	0.5955	0.0001	0.0013	0.0037
12	anthurium	0	0.5821	0	0.0001	0.0004
13	anthurium	0	0.7629	0	0.0001	0.0005
14	rose	0.0001	0.1753	0.0019	0.0043	0.0052
15	artichoke	0.0001	0	0.9667	0	0
16	artichoke	0.0001	0	0.9639	0	0.0001
17	artichoke	0.0001	0	0.9819	0	0.0002
18	artichoke	0.0001	0	0.9981	0	0
19	artichoke	0.0022	0	0.9830	0	0.0002
20	artichoke	0	0	0.9892	0	0
21	artichoke	0.0009	0	0.8895	0	0
22	artichoke	0.0114	0	0.7085	0	0
23	azalea	0	0	0	0.8115	0
24	azalea	0	0.0001	0	0.5035	0.0003
25	azalea	0.0002	0.0001	0.0001	0.4785	0.0004
26	azalea	0.0001	0	0	0.3956	0.0001
27	azalea	0.0002	0	0	0.2510	0.0013
28	azalea	0.0002	0	0.0002	0.2798	0.0009
29	azalea	0	0	0	0.3007	0.0004
30	azalea	0	0	0	0.2501	0.0001
31	azalea	0	0.0001	0	0.3576	0.0001
32	azalea	0	0	0	0.5040	0
33	cyclamen	0.0001	0.0043	0	0.0042	0.0013
34	ball moss	0.0017	0.0066	0.0007	0.0046	0.3899
35	ball moss	0.0010	0.0025	0.0001	0.0056	0.3591
36	ball moss	0.0020	0.0013	0.0270	0.0015	0.4201

	predicted	alpine sea holly	anthurium	artichoke	azalea	ball moss
37	ball moss	0.0036	0.0025	0.0610	0.0011	0.3203
38	balloon flower	0	0	0	0.0001	0
39	stemless gentian	0.0001	0.0012	0.0022	0.0005	0.0198
40	morning glory	0.0002	0.0001	0	0.0004	0.0017
41	balloon flower	0	0	0	0.0037	0.0001
42	balloon flower	0.0002	0	0	0.0032	0.0003
43	barbeton daisy	0	0	0	0	0
44	mexican aster	0.0001	0	0	0	0
45	purple coneflower	0.0025	0.0002	0.0004	0.0049	0.0355
46	barbeton daisy	0.0001	0	0	0	0
47	barbeton daisy	0	0	0	0	0
48	barbeton daisy	0	0	0	0	0
49	barbeton daisy	0.0071	0	0.0002	0.0189	0.0006
50	barbeton daisy	0	0	0	0.0001	0
51	barbeton daisy	0	0	0	0	0
52	barbeton daisy	0	0	0	0	0
53	barbeton daisy	0	0	0	0	0
54	barbeton daisy	0.0005	0	0	0.0049	0.0001
55	barbeton daisy	0	0	0	0	0
56	globe-flower	0	0.0001	0	0.0001	0
57	bearded iris	0	0	0	0.0121	0.0004
58	mexican petunia	0.0002	0	0.0001	0.1155	0.0020
59	mexican petunia	0.0001	0	0.0002	0.1324	0.0011
60	mexican petunia	0.0001	0	0.0008	0.0169	0.0004
61	bee balm	0.0055	0	0.0006	0.0090	0.0023
62	bee balm	0.0041	0.0001	0.0003	0.0034	0.0018
63	bee balm	0.0139	0	0.0136	0.0053	0.0030
64	bee balm	0.0309	0	0.0052	0.0023	0.0019
65	bee balm	0.0170	0	0.0004	0.0685	0.0039
66	great masterwort	0.0149	0	0.0574	0.0010	0.0013
67	bee balm	0.0063	0	0.1641	0.0019	0.0031
68	bird of paradise	0	0.0002	0	0	0.0001
69	bird of paradise	0	0	0	0	0.0001

	predicted	alpine sea holly	anthurium	artichoke	azalea	ball moss
70	bird of paradise	0	0.0008	0	0	0.0007
71	bird of paradise	0.0013	0.0391	0.0001	0	0.0640
72	bird of paradise	0	0.0001	0	0	0.0002
73	bird of paradise	0.0003	0.0025	0	0	0.0006
74	bird of paradise	0	0.0001	0	0	0
75	bird of paradise	0.0001	0.0003	0	0	0.0002
76	bishop of lland...	0	0	0	0.0001	0
77	bishop of lland...	0	0	0	0.0002	0
78	bishop of lland...	0.0001	0	0	0.0006	0.0002
79	bishop of lland...	0.0004	0	0	0.0019	0.0001
80	bishop of lland...	0	0	0	0.0015	0
81	bishop of lland...	0	0	0	0.0010	0
82	bishop of lland...	0	0	0	0.0001	0
83	bishop of lland...	0	0	0	0.0011	0
84	bishop of lland...	0	0	0	0.0009	0
85	canna lily	0.0002	0.0021	0.0001	0.0118	0.0002
86	bishop of lland...	0.0001	0	0	0.0025	0.0002
87	black-eyed susan	0	0	0	0	0
88	black-eyed susan	0	0	0	0.0004	0
89	black-eyed susan	0	0	0	0.0001	0
90	black-eyed susan	0.0001	0	0	0.0005	0
91	black-eyed susan	0	0	0	0.0003	0
92	blackberry lily	0	0	0	0	0
93	blackberry lily	0	0	0	0	0
94	blackberry lily	0.0001	0	0	0	0
95	blackberry lily	0	0	0	0	0
96	blackberry lily	0.0001	0.0004	0.0001	0.0008	0.0004
97	blanket flower	0.0008	0	0.0002	0.0020	0
98	blanket flower	0.0004	0	0.0002	0.0052	0.0008
99	blanket flower	0.0027	0	0.0004	0.0001	0
100	blanket flower	0.0028	0	0.0002	0.0003	0

⋮

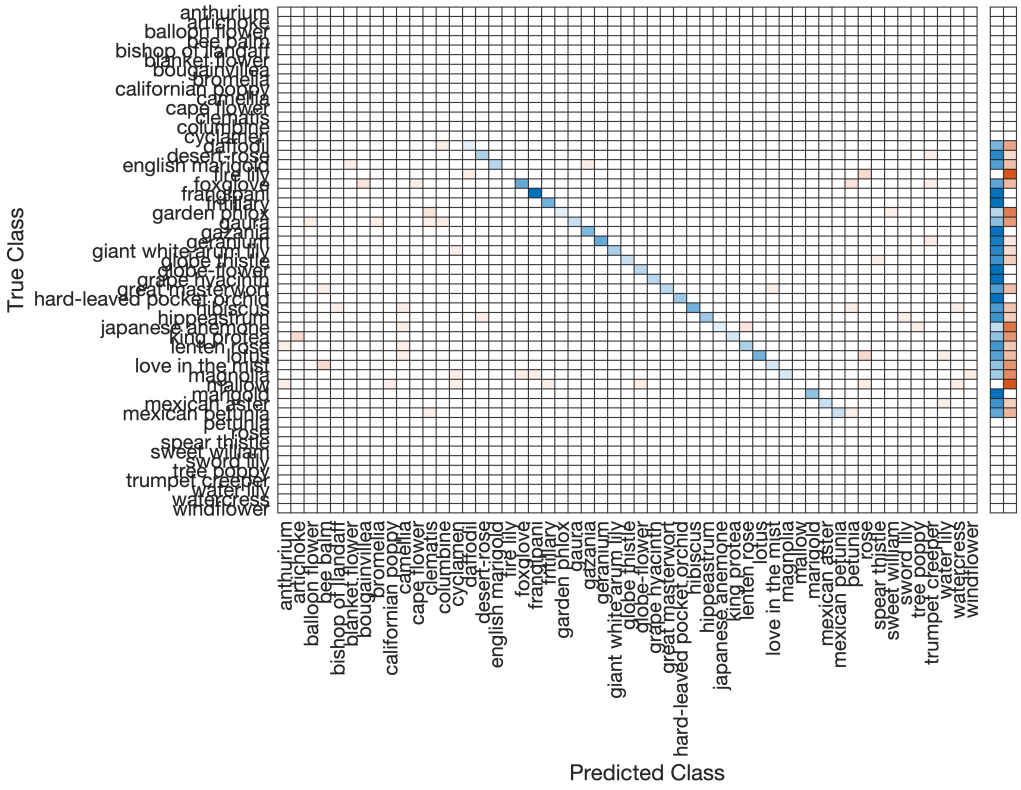
```
nnz(predicted~=valset.response)/length(predicted)
```

```
ans = 0.2745
```

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.

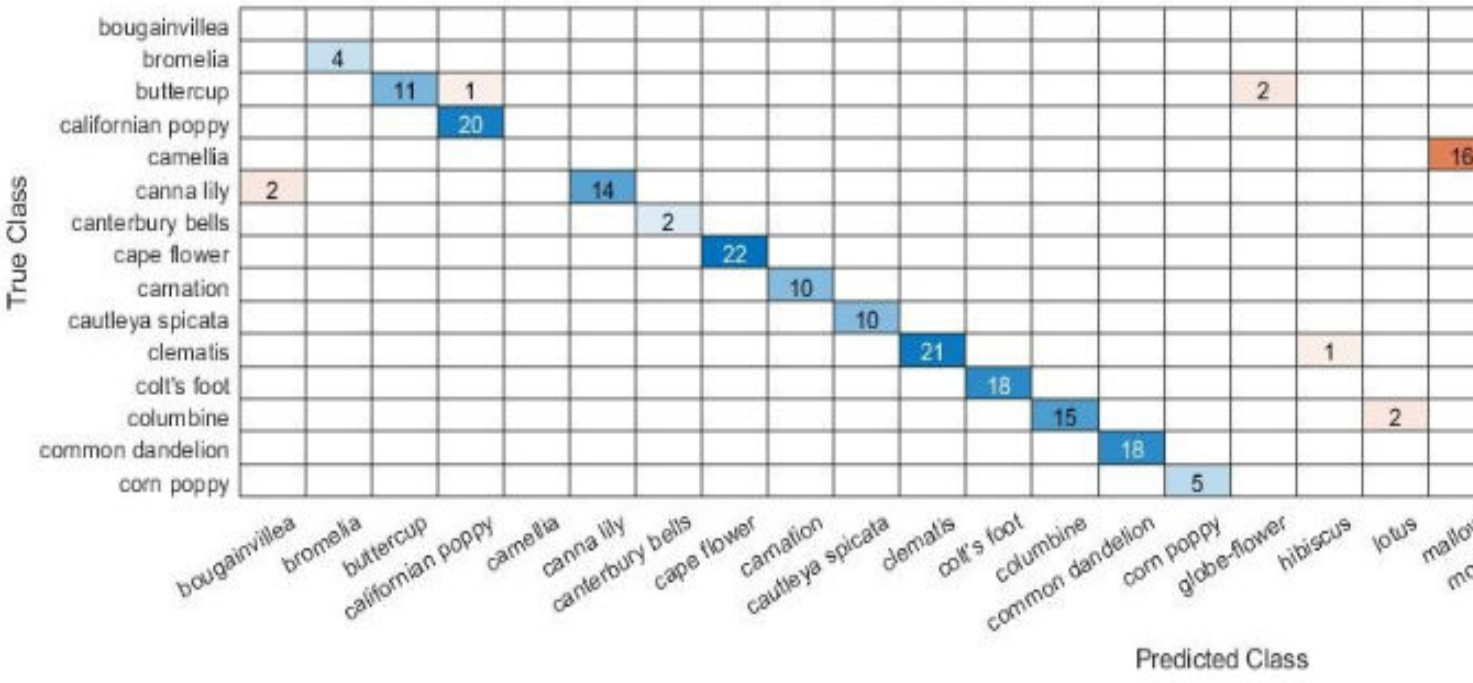
```
cm2 = confusionchart(string(valset.response(245:450)),string(predicted(245:450)),"RowSu
```



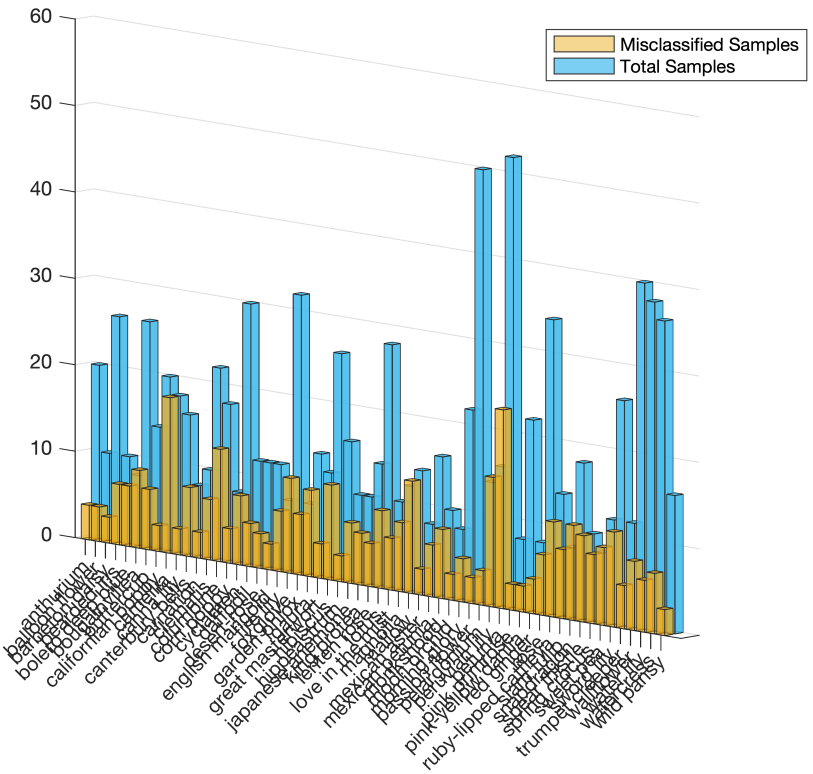

```
cm2 =
  ConfusionMatrixChart with properties:

    NormalizedValues: [53×53 double]
         ClassLabels: [53×1 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 == "camellia");
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);
size(acs1)
```

```
ans = 1×3
     1     1   102
```

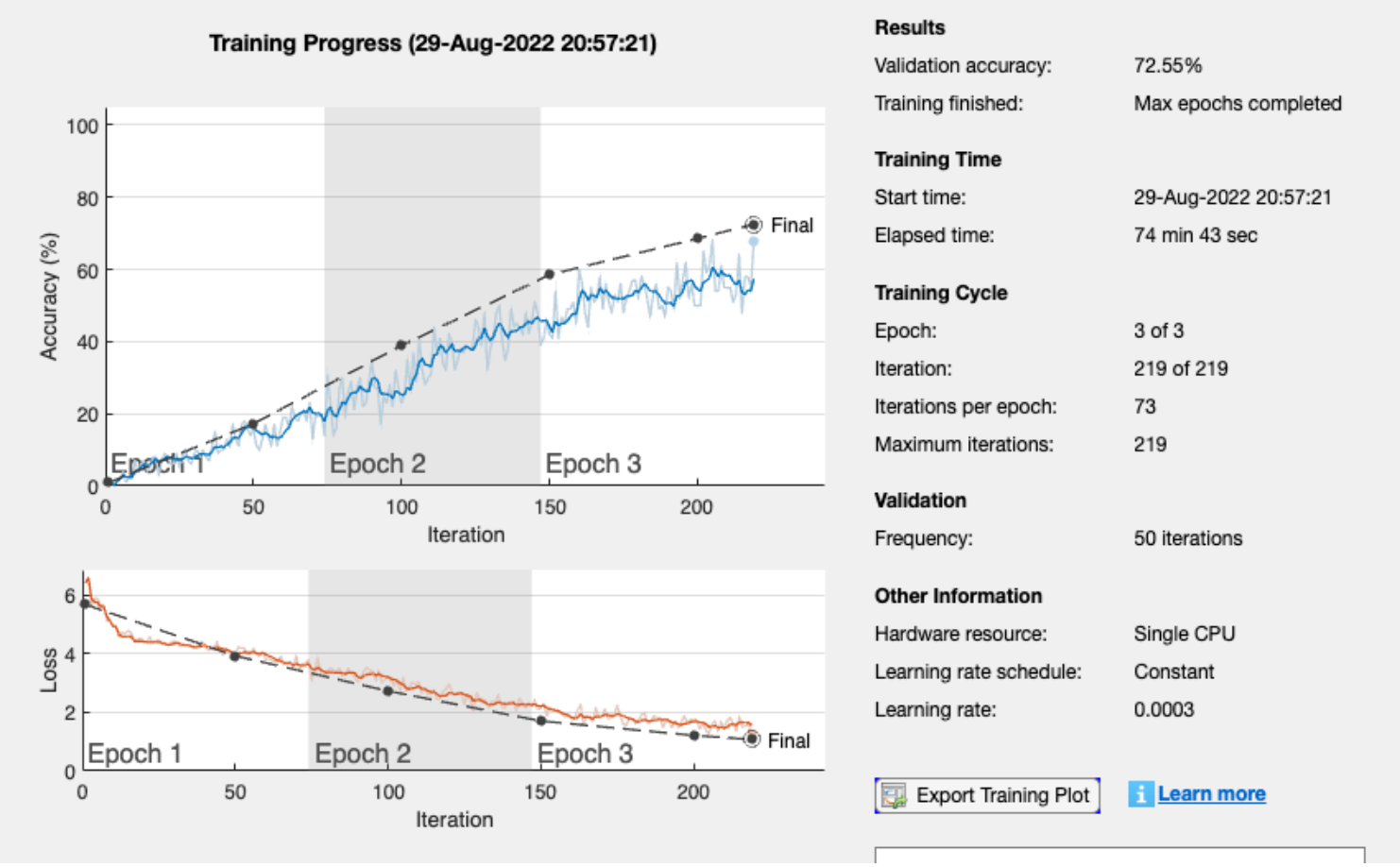
```
imshow(imresize(imtile(rescale(acs1)),[1000 1000]))
```

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.

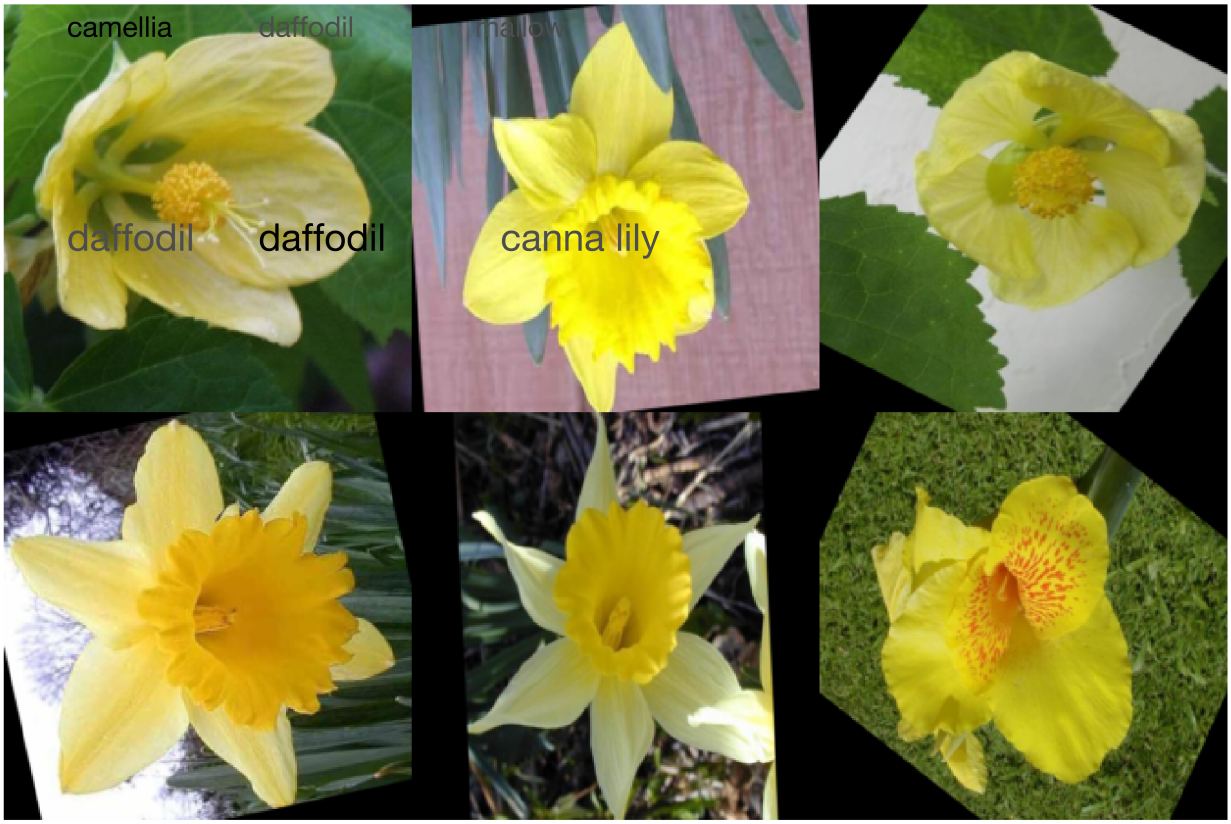
```
layer = "fc";
ind = find(valset.response == "camellia");
N = ind(6);
acs = activations(FlowerNet,auds,layer,"MiniBatchSize",1,"OutputAs","rows");
acs1 = activations(FlowerNet,valset(N,:),layer,"OutputAs","rows");
```

```
[idx, d] = knnsearch(acs, acs1, 'K', 5,"NSMethod","kdtree");   %machine learning algor
%idx(1) = [];
searchresults = readByIndex(auds,idx);
```

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




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

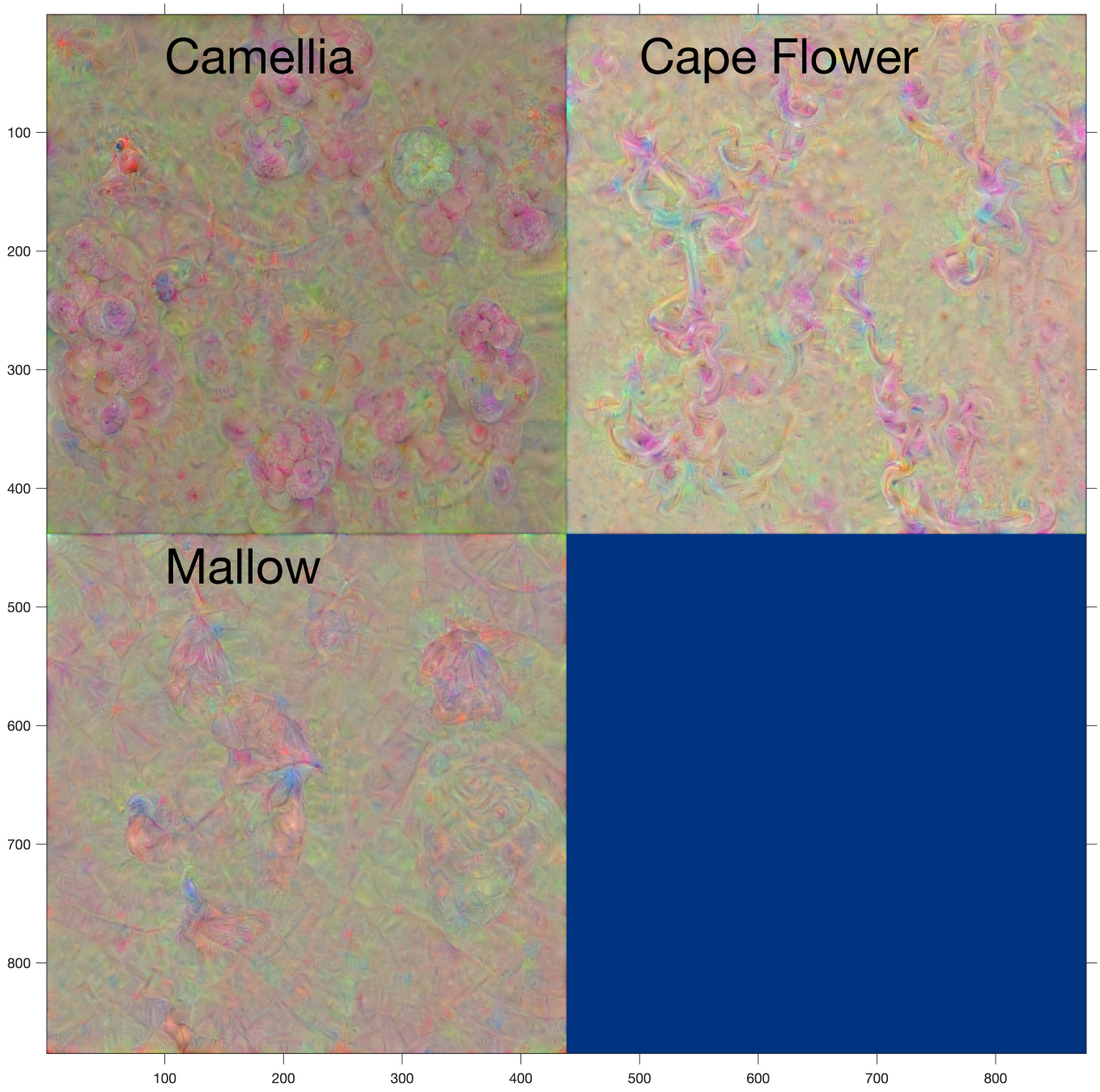


Using Deep Dream Image

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 57],"Verbose",false,"NumIterations",10,"Exe
DD = deepDreamImage(FlowerNet,'fc',[20 23 57],"Verbose",false,"NumIterations",10); %20
```

```
imshow(imtile(DD(:,:,:,1:3),"BackgroundColor", [0 0.2 0.5]))
axis on
text(100,35,'Camellia',"FontSize",35); text(500,35,'Cape Flower',"FontSize",35); text(
```



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.respo
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,
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
```