Final Project: Cats vs. Dogs

Department of Computer & Electrical Engineering and Computer Science
Introduction to Artificial Intelligence – CAP 4630

Tsz Shing Tsoi

Course Instructor
Dr. Oge Marques

Monday, July 22, 2019

Table of Contents

Abstract

This Final Project: Cats vs. Dogs implemented deep learning solutions for classifying colored images of cats and dogs using the dataset available from Kaggle (https://www.kaggle.com/c/dogs-vs-cats/data). This dataset was used in the Kaggle challenge which ran from 2013 to 2014; and the winner, Pierre Sermanet, achieved a classification accuracy of 98.914%. Through transfer learning with a pretrained convolutional neural network, this final project developed and evaluated three deep learning classifiers: a baseline classifier and two improved classifiers.

As stated in the guidelines for this final project provided by Dr. Marques, the goal of this final project is to learn how to implement complete and fully functional deep learning solutions for image classification problems, to use neural networks under transfer learning, and to fine-tune and evaluate different deep learning solutions to the same problem. The MATLAB program used in this final project is based on the MATLAB starter code "`final_project_starter_cap4630.m`" and image processing function "`readAndPreprocessImage.m`" provided by Dr. Marques.

Dataset

The dataset used in this final project includes 25,000 labeled images of cats and dogs in the training data and 12,500 unlabeled images in the test data. Each image is a 24-bit 3-channel (red, green and blue) image with varying dimensions. The file names in the training data are labeled as cat.XX.jpg and dog.XX.jpg while the file names in the test data are simply XX.jpg, where XX is the image sequence number.

These images are easily recognized by human, although the image composition varies greatly between photos with different postures, background, lighting conditions, angles, and number of pets. However, a few images are neither cats nor dogs. A small subset of sample images is shown in Exhibit 1.

Transfer Learning

Transfer learning was used to implement the deep learning solution in this final project. Transfer learning uses a pretrained neural network (NN) developed for similar tasks, which would reduce the errors made by new learners of NN and the time needed on training to achieve good performance. Convolutional neural network (CNN) has been shown to be particularly effective in image classification tasks. Therefore, the pretrained NN used in this final project is a CNN, AlexNet, which has been trained on approximately 1.2 million images from the ImageNet Dataset (http://image-net.org/index) with 1,000 object categories. AlexNet is available from the Deep Learning Toolbox Model for AlexNet Network in MATLAB.

AlexNet has 25 layers, including the input layer of a 3-channel image of 227 by 227 pixels and the output layer which has 1,000 classes. The network includes 5 convolutional layers and the activation function is rectified linear unit (ReLU), i.e. $f(z) \equiv \max(0, z)$, with 3 by 3 max-pooling layers. 3 fully connected layers with 50% dropout are used after the convolutional layers. Table 1 illustrates the AlexNet layers.

The modified CNN used in this final project is based on the pretrained AlexNet with the last fully connected layer replaced by a 2-node fully connected layer and the subsequent softmax and classification output layers replaced accordingly.

Exhibit 1 Sample Images

Data Preprocessing

Data preprocessing is typically minimal for neural networks. In this final project, data preprocessing involved importing the labeled image dataset as an `ImageDatastore` object, and equalizing the number of images in each class. The `ReadFcn` property of the `ImageDatastore` object was set to the `readAndPreprocessImage()` function to re-size each image to a 3-channel image of 227 by 227 pixels, which is required in the input layer of the CNN, as each image is processed.

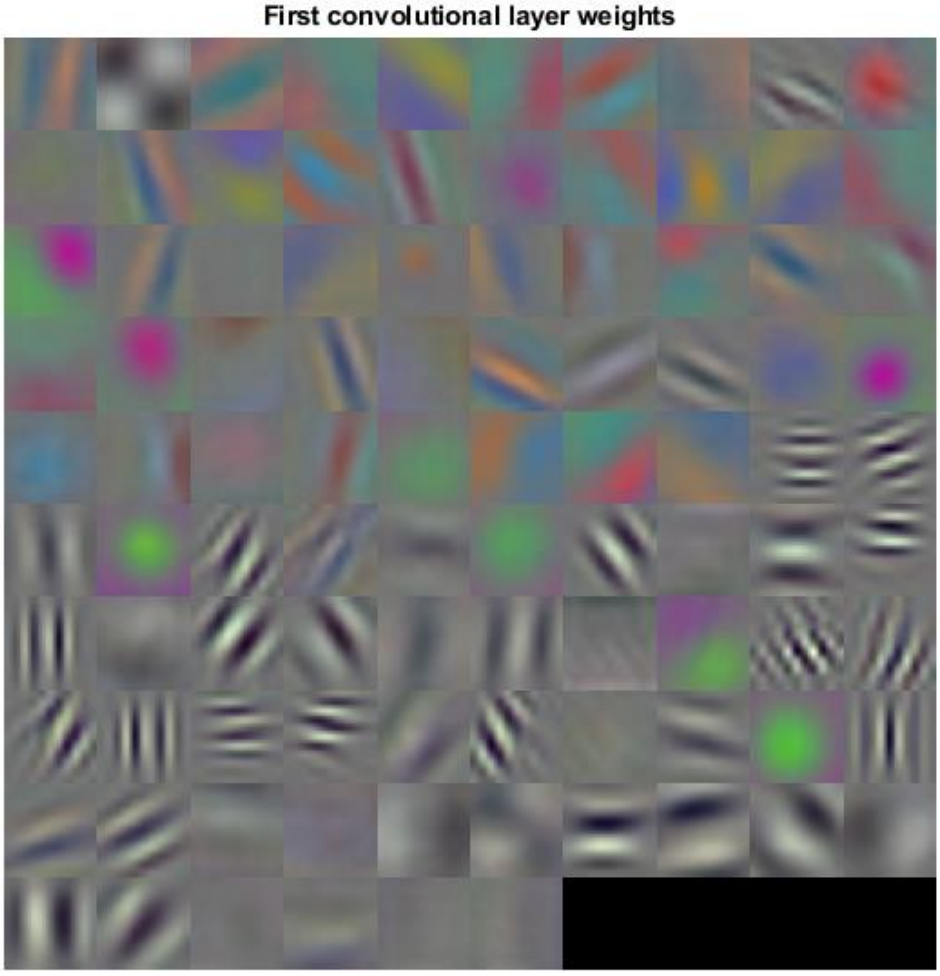
Table 1 AlexNet Layers

#	Name	Type	Description
1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Grouped Convolution	2 groups of 128 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Grouped Convolution	2 groups of 192 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Grouped Convolution	2 groups of 128 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench' and 999 other classes

Warm-Up Exercise

The warm-up exercise introduces the workflow in deep learning solutions. It provides guidance to inspect the AlexNet layers, and to display the weights of the 96 feature maps of 11 by 11 pixels by 3 channels in the first convolutional layer (layer #2 in Table 1) as a montage of images, as shown in Figure 1. This exercise also imported and pre-processed a very small subset of the training data (20 images of cats and 20 images of dogs) to train the modified CNN. 28 (70%) of these 40 images were used in training and the remaining 12 images (30%) were used for validation.

Figure 1 Illustration of First Convolutional Layer Weights



Two deep learning warm-up exercises were performed. The warm-up exercise #1 trained the CNN with the training options shown in Table 2, and achieved a validation accuracy of 83.33%. This trained CNN is able to correctly classify unseen images of a dog (dog.jpg) and a cat (cat.jpg) with near perfect confidence, but is unable to correctly classify the iconic Doge (doge.jpg).

Table 2 CNN Training Options – Warm-Up Exercise

Options	Settings – Warm-Up Exercise #1	Settings – Warm-Up Exercise #2 with Data Augmentation
MiniBatchSize	10	10
MaxEpochs	6	8
InitialLearnRate	0.0001	0.0003
Shuffle	'every-epoch'	'once' (default)
ValidationFrequency	3	50 (default)

The warm-up exercise #2 trained the CNN with data augmentation using the training options also shown in Table 2. Data augmentation is a preprocessing step and a technique to prevent the network from overfitting by randomly perturbing the training images. However, it does not increase the actual number of images used in training. Several types of image processing techniques were used in this exercise, including:

1. Randomly flip each image horizontally (mirror image)
2. Randomly translate each image vertically and horizontally by up to 30 pixels in each direction
3. Randomly scale the length and width of each image by a factor between 0.9 and 1.1

The validation accuracy is 75.00%. This is worse than the model without data augmentation, which could be attributed to different training options used and the very low number of training samples. Since the `MiniBatchSize` of 10 does not evenly divide the training sample of 28 images, the same 8 images would be discarded in every epoch, effectively reducing the sample size to 20 images in this second exercise. Nevertheless, the validation accuracies of both networks are far from the accuracy of 98.916% achieved by the Kaggle challenge winner. This could largely be explained by the limited training samples.

The last part of the warm-up exercise trained the modified CNN with the full Kaggle training dataset. This becomes the baseline classifier in the final solution and is discussed next.

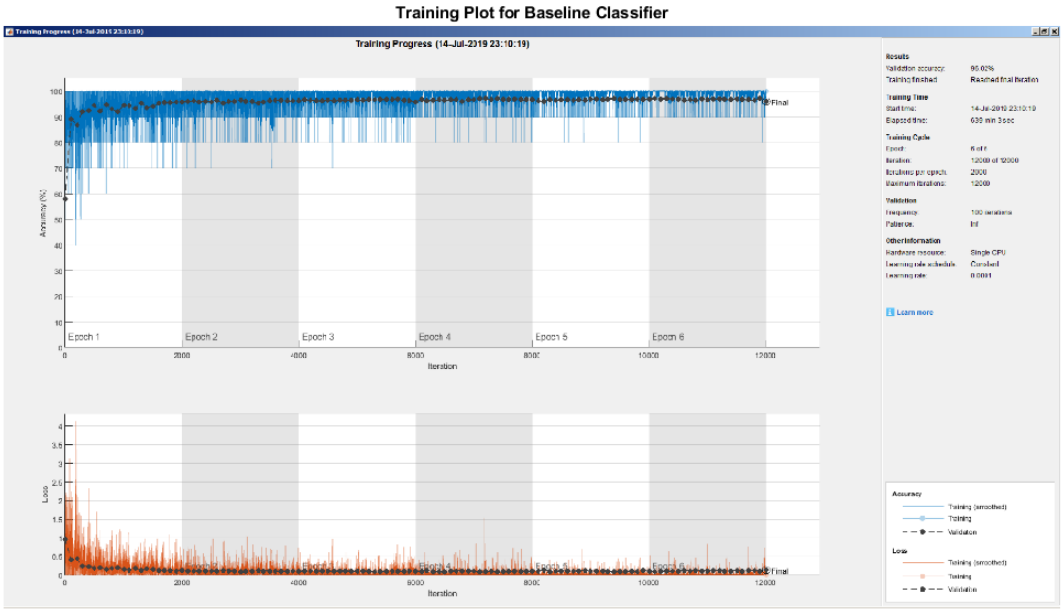
Baseline Classifier

The baseline classifier follows very similar workflow as in the warm-up exercise #1, but with the full Kaggle training dataset of 25,000 images, partitioned with 20,000 images (80%) used for training and 5,000 images (20%) used for hold-out validation. The training options shown in Table 3 are the same as in the warm-up exercise #1 except the validation frequency was increased to 100 to increase the training speed. The training plot is shown in Figure 2 with a validation accuracy of 96.02%

Table 3 CNN Training Options – Baseline Classifier

Options	Settings – Baseline Classifier
MiniBatchSize	10
MaxEpochs	6
InitialLearnRate	0.0001
Shuffle	‘every-epoch’
ValidationFrequency	100

Figure 2 Training Plot – Baseline Classifier



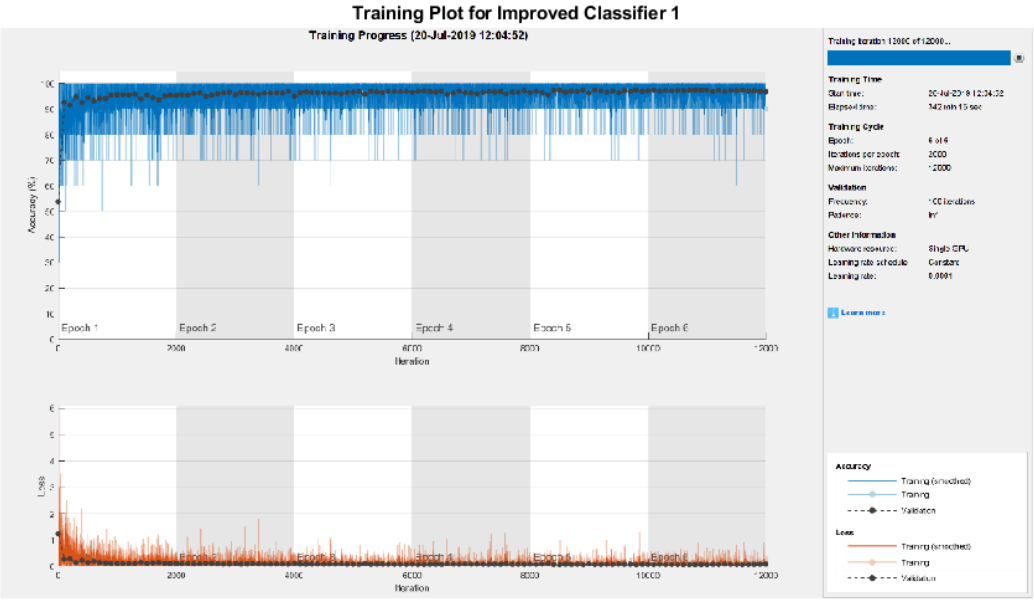
Improved Classifier 1

The improved classifier 1 follows very similar workflow as in the warm-up exercise #2 with data augmentation using the full Kaggle training dataset. The dataset partition is the same as in the baseline classifier, i.e. 80% for training and 20% for hold-out validation. For comparison purpose, the training options also shown in Table 4 are the same as in the baseline classifier. The training plot is shown in Figure 3. Improved classifier 1 improves the validation accuracy to 96.72%.

Table 4 CNN Training Options – Improved Classifier 1

Options	Settings – Improved Classifier 1
MiniBatchSize	10
MaxEpochs	6
InitialLearnRate	0.0001
Shuffle	'every-epoch'
ValidationFrequency	100

Figure 3 Training Plot – Improved Classifier 1



Improved Classifier 2

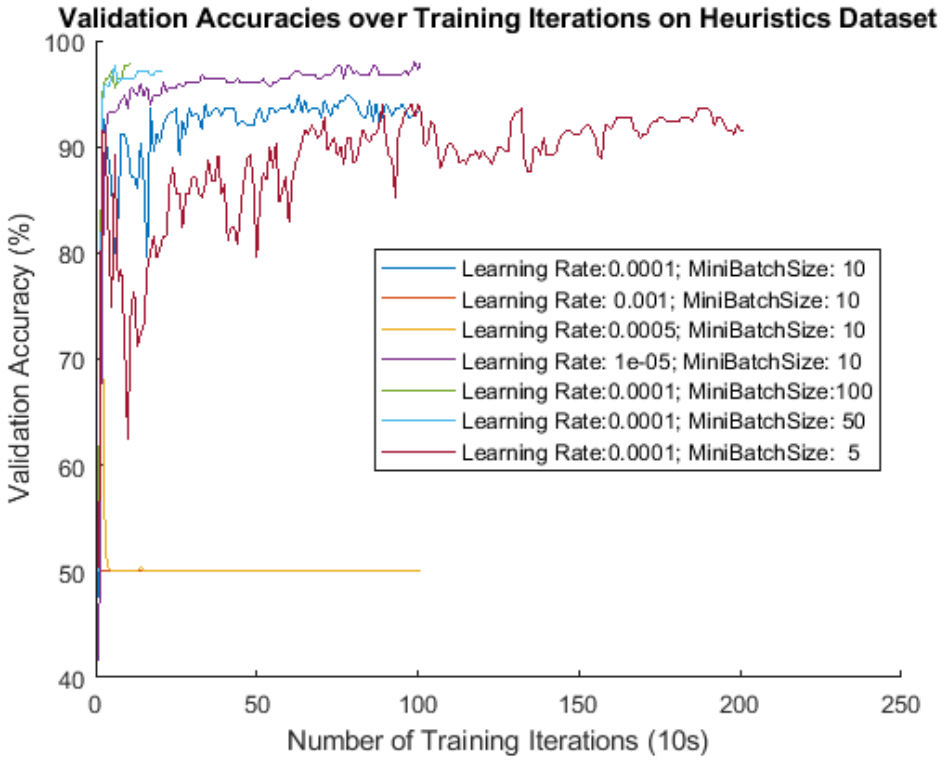
The improved classifier 2 used a different set of hyperparameters to improve the performance. The target hyperparameters include `MiniBatchSize`, `MaxEpochs` and `InitialLearnRate`, with the potential to implement a learning rate schedule. The set of preferred hyperparameters was determined by trial-and-error using heuristics, with one hyperparameter change at a time. The heuristics is a reduced dataset consisting 5% (1,000 images) of the training data and 5% (250 images) of the hold-out validation data.

After partitioning the heuristics dataset, the set of hyperparameters shown in Table 5 was identified for trial-and-error. The first set of hyperparameters is identical to the baseline classifier and is a control set. 3 different values of `MiniBatchSize` and 3 different values of `MiniBatchSize` were identified. The `MaxEpochs` was increased to 10 for all scenarios to account for decreased learning rate and increased `MiniBatchSize` in some scenarios. The random generator was re-seeded before each training scenario so that the randomness factor was controlled. The final validation accuracy is also shown in Table 5. The plot of validation accuracy over training iterations is shown in Figure 4.

Table 5 Scenarios of Hyperparameters and Final Validation Accuracy

Scenario	Learning Rate	Mini Batch Size	Final Validation Accuracy
1 (Control)	0.0001	10	93.6%
2	0.001	10	50%
3	0.0005	10	50%
4	0.00001	10	98%
5	0.0001	100	98%
6	0.0001	50	97.2%
7	0.0001	5	91.6%

Figure 4 Validation Accuracy Plot on Hyperparameter Scenarios



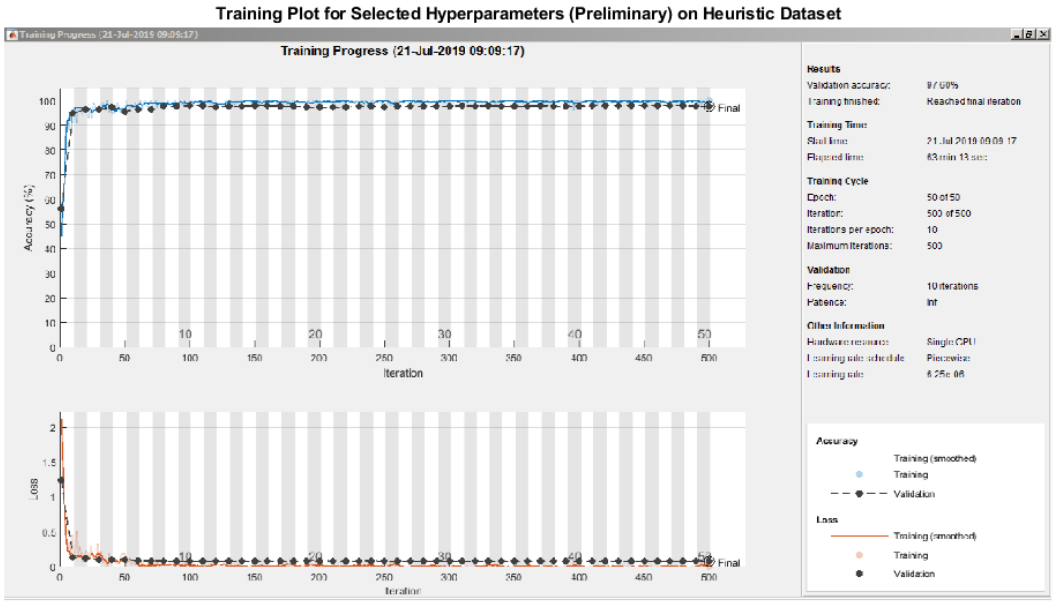
As shown in Table 5, a low learning rate (0.00001) and a high mini batch size (100) appear to result in the best performance. It is also noted that a learning rate equal to or greater than 0.0005 essentially results in random guess (50% accuracy) as the step-size is too large and the loss function fails to converge to a minimum.

Figure 4 shows that the validation accuracies for these two scenarios are on the increasing trend prior to the end of the training. Therefore, using a learning rate schedule and increasing the number of epochs may help speed up the training and improve the performance further. The preliminary settings shown in Table 6 were used to train with the heuristics dataset and achieved a validation accuracy of 97.6%. The training plot is shown in Figure 5.

Table 6 CNN Training Options – Preferred Scenario (Preliminary)

Options	Preliminary Settings – Preferred Scenario
MiniBatchSize	100
MaxEpochs	50
InitialLearnRate	0.0001
LearnRateSchedule	'piecewise'
LearnRateDropPeriod	10
LearnRateDropFactor	0.5
Shuffle	'every-epoch'
ValidationFrequency	10

Figure 5 Training Plot – Preferred Hyperparameter Scenario (Preliminary)



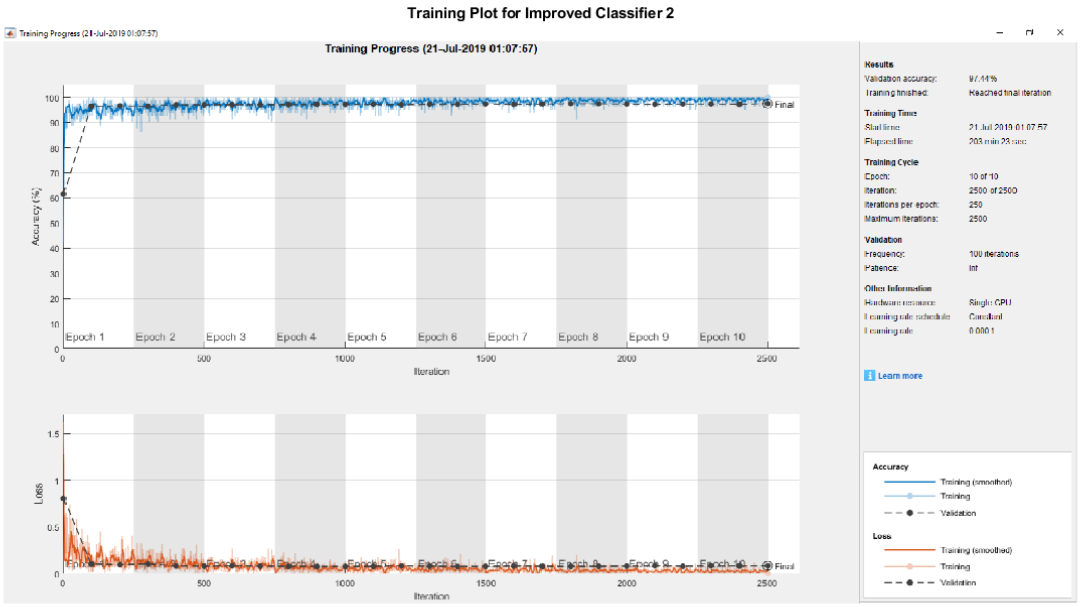
The training plot shown in Figure 5 shows that the validation accuracy remains relatively constant after approximately 10 epochs and a learning rate lower than 0.0001 does not appear to improve the performance further. Therefore, a single learning rate of 0.0001 and 10 training epochs were selected in the final settings for improved classifier 2, as shown in Table 7. The training plot for improved classifier 2 is shown in Figure 6 with a final validation accuracy of 97.44%.

Table 7 CNN Training Options – Improved Classifier 2

Options	Settings – Improved Classifier 2
MiniBatchSize	80*
MaxEpochs	10
InitialLearnRate	0.0001
LearnRateSchedule	'none'
LearnRateDropPeriod	N/A
LearnRateDropFactor	N/A
Shuffle	'every-epoch'
ValidationFrequency	100

* A reduced `MiniBatchSize` was used due to the memory limitation of the GPU on hand at the time of training.

Figure 6 Training Plot – Improved Classifier 2



Performance Evaluation

The 3 trained CNN classifiers were used to classify the Kaggle test dataset. Since the test dataset was unlabeled, the dataset was first classified with an ensemble of the three classifiers developed above, and then manually corrected and re-labeled. Table 8 summarizes the accuracy on the test set. Detailed evaluation of each classifier is discussed next. All detailed evaluations were performed on the test set.

Table 8 Model Accuracy Summary

Model Name	Accuracy on Validation Data in Training Set	Accuracy on Test Set
Baseline Classifier	96.02%	96.81%
Improved Classifier 1	96.72%	96.93%
Improved Classifier 2	97.44%	97.70%
Ensemble of the 3 Classifiers	Not Evaluated	97.88%

Baseline Classifier

The confusion matrix for the test dataset is shown in Figure 7, and the receiver operating characteristic (ROC) curve is plotted and shown in Figure 8. The area under the curve (AUC) is computed to be 0.9964. Samples of correctly (with the highest confidence) and incorrectly (with the lowest confidence) classified images are shown in Exhibit 2 and Exhibit 3 respectively.

Figure 7 **Test Dataset Confusion Matrix – Baseline Classifier**

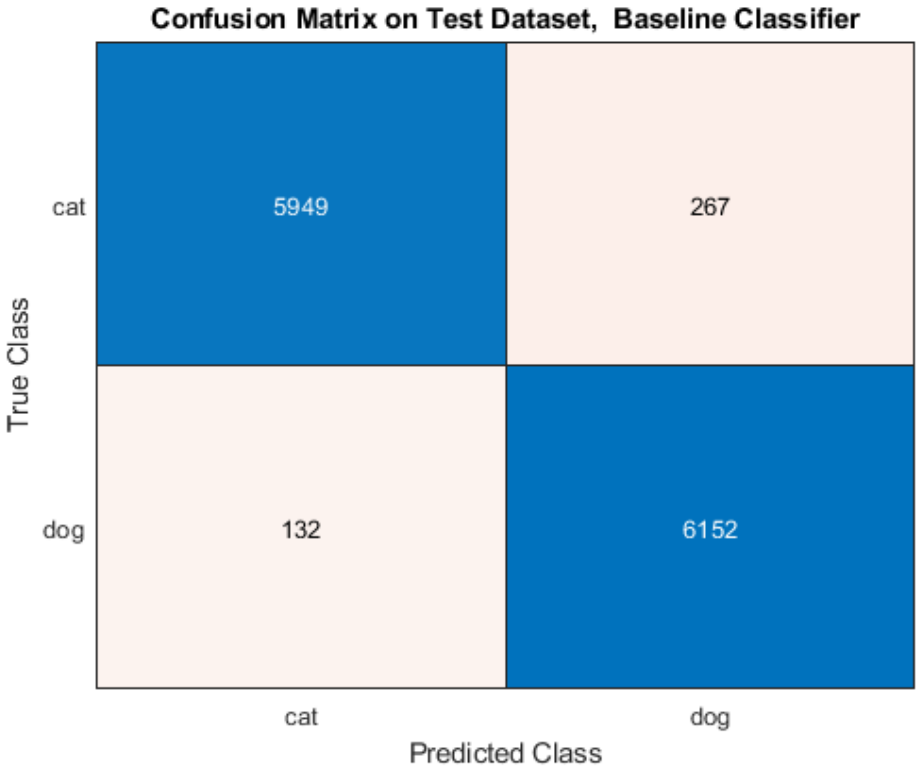


Figure 8 **ROC Curve on Test Dataset – Baseline Classifier**

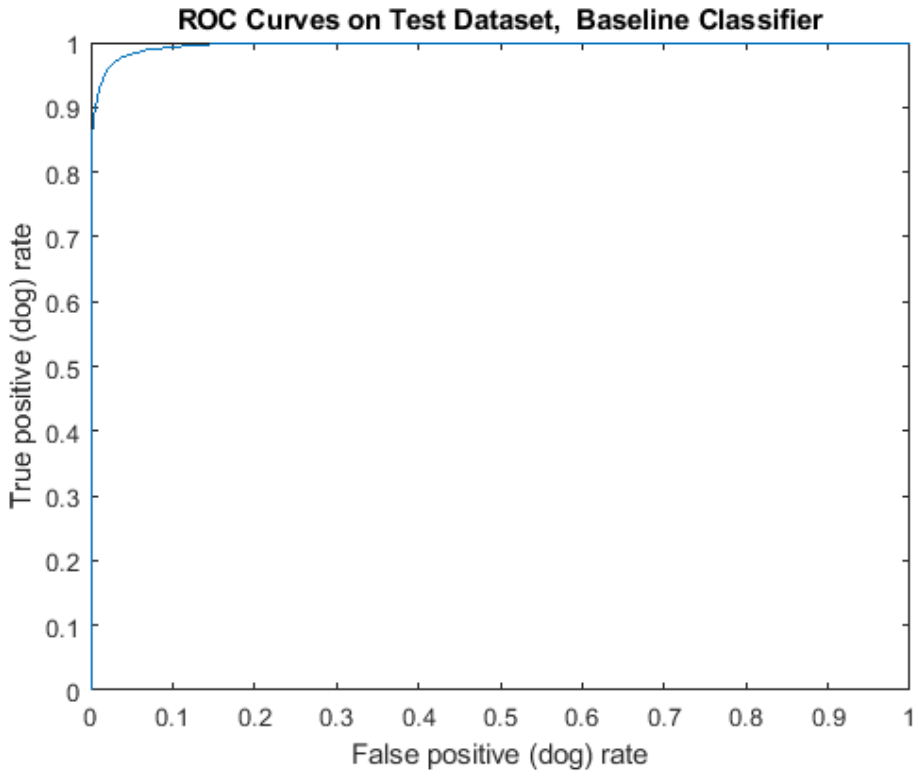


Exhibit 2 Samples of Correctly Classified Images – Baseline Classifier

Sample - Correctly Classified as Cats, Baseline Classifier | Confidence: 100.00%

Sample - Correctly Classified as Dogs, Baseline Classifier | Confidence: 100.00%

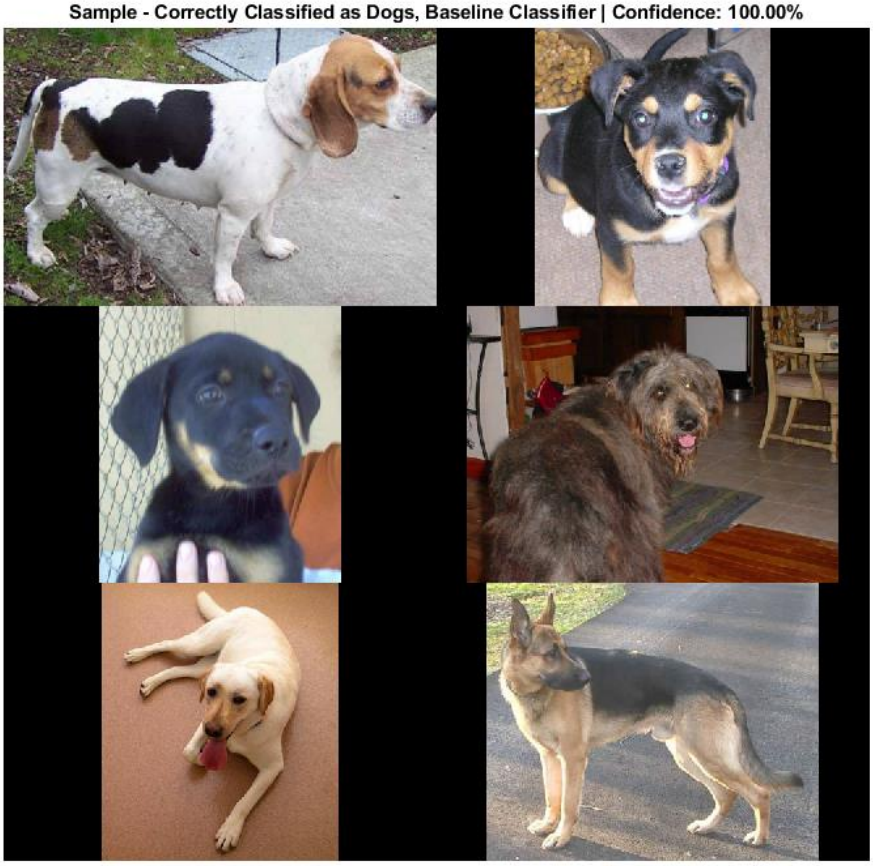
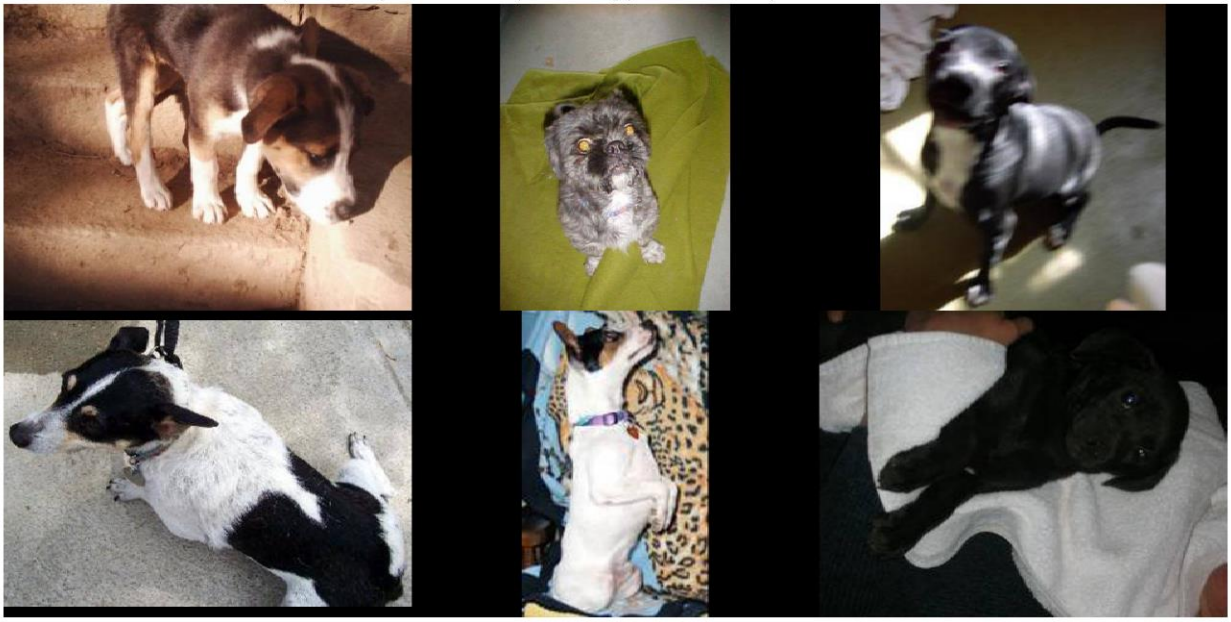


Exhibit 3 Samples of Incorrectly Classified Images – Baseline Classifier

Sample - Incorrectly Classified as Dogs (Should be Cats), Baseline Classifier | Confidence: 50.07% to 50.76%

Sample - Incorrectly Classified as Cats (Should be Dogs), Baseline Classifier | Confidence: 51.03% to 52.91%



Improved Classifier 1

The confusion matrix for the test dataset is shown in Figure 9, and the ROC curve is plotted and shown in Figure 10. The AUC is computed to be 0.9963. Samples of correctly (with the highest confidence) and incorrectly (with the lowest confidence) classified images are shown in Exhibit 4 and Exhibit 5 respectively.

Figure 9 Test Dataset Confusion Matrix – Improved Classifier 1

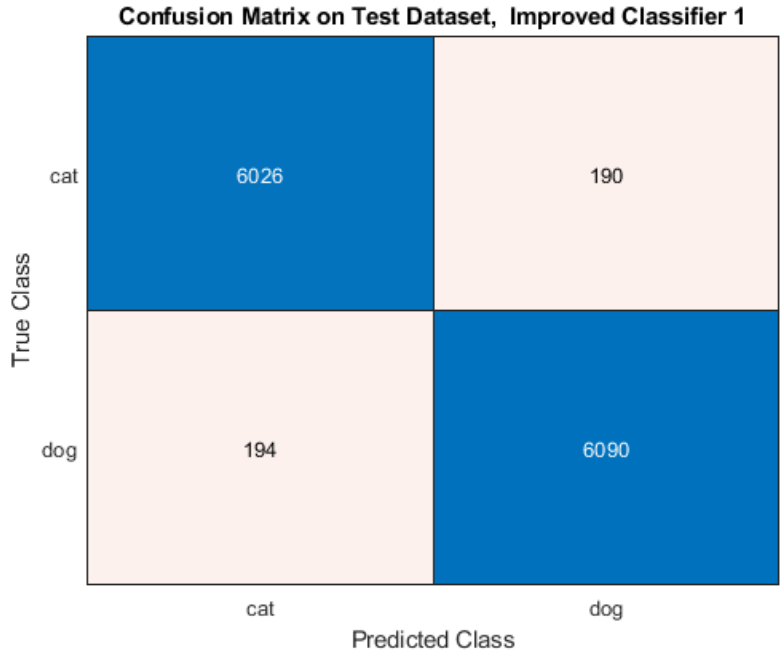


Figure 10 ROC Curve on Test Dataset – Improved Classifier 1

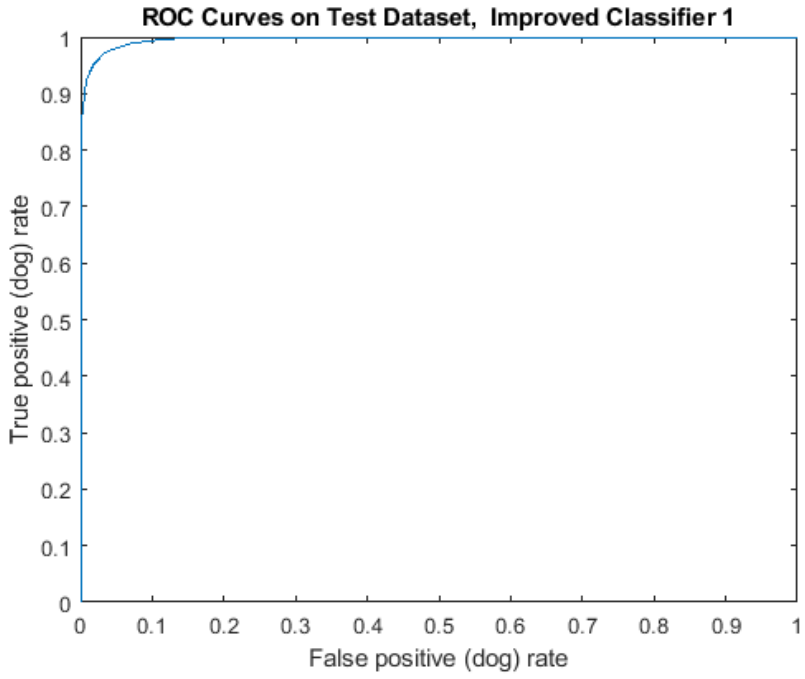
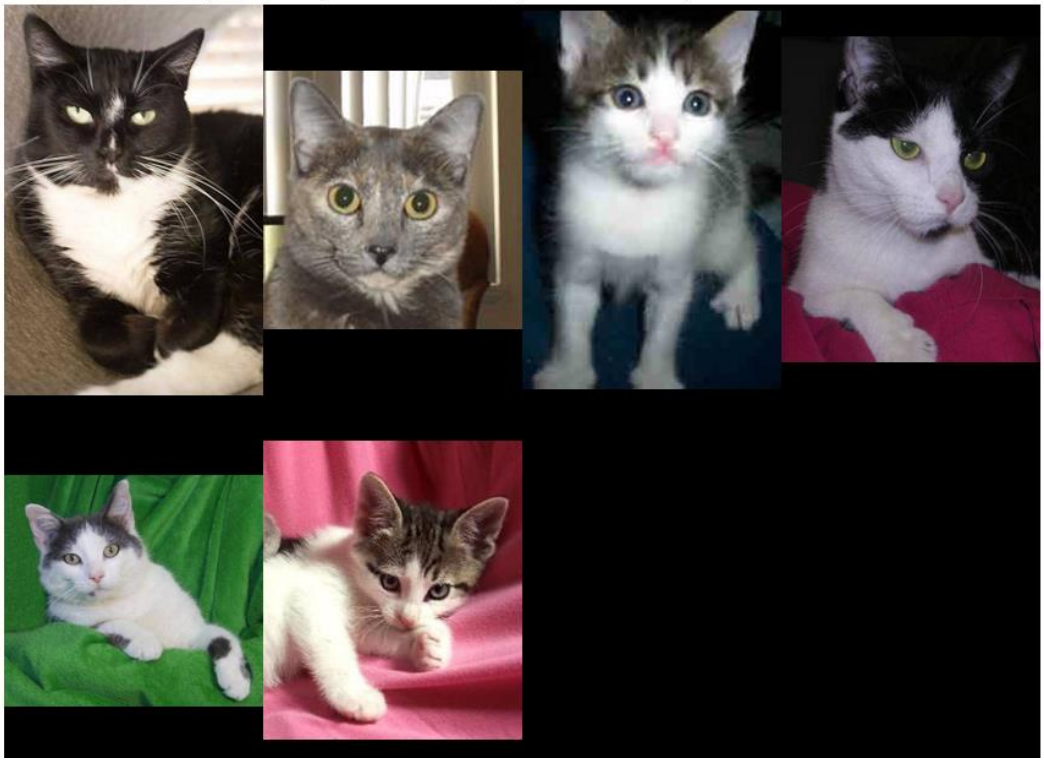


Exhibit 4 Samples of Correctly Classified Images – Improved Classifier 1

Sample - Correctly Classified as Cats, Improved Classifier 1 | Confidence: 100.00%



Sample - Correctly Classified as Dogs, Improved Classifier 1 | Confidence: 100.00%

Exhibit 5 Samples of Incorrectly Classified Images – Improved Classifier 1

Sample - Incorrectly Classified as Dogs (Should be Cats), Improved Classifier 1 | Confidence: 50.20% to 52.41%

Sample - Incorrectly Classified as Cats (Should be Dogs), Improved Classifier 1 | Confidence: 50.31% to 50.92%

Improved Classifier 2

The confusion matrix for the test dataset is shown in Figure 11, and the ROC curve is plotted and shown in Figure 12. The AUC is computed to be 0.9978. Samples of correctly (with the highest confidence) and incorrectly (with the lowest confidence) classified images are shown in Exhibit 6 and Exhibit 7 respectively.

Figure 11 Test Dataset Confusion Matrix – Improved Classifier 2

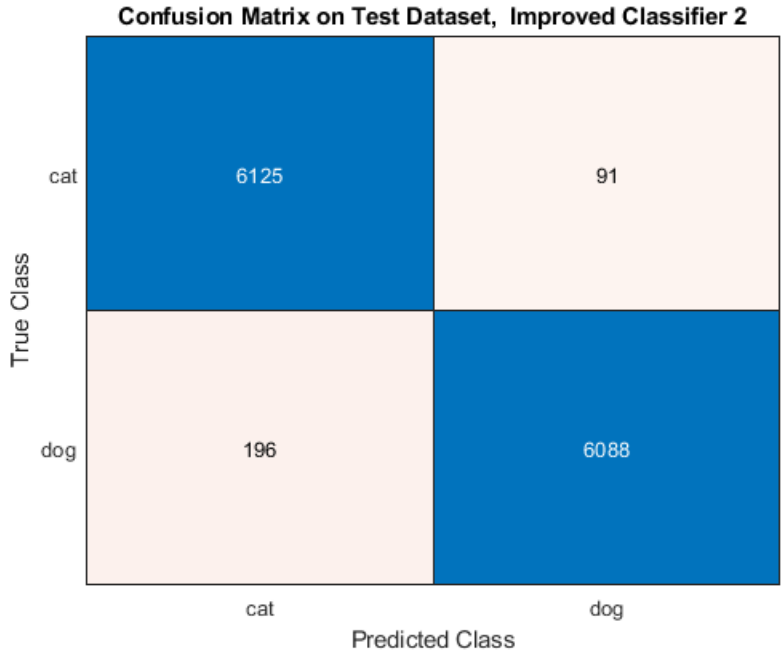


Figure 12 ROC Curve on Test Dataset – Improved Classifier 2

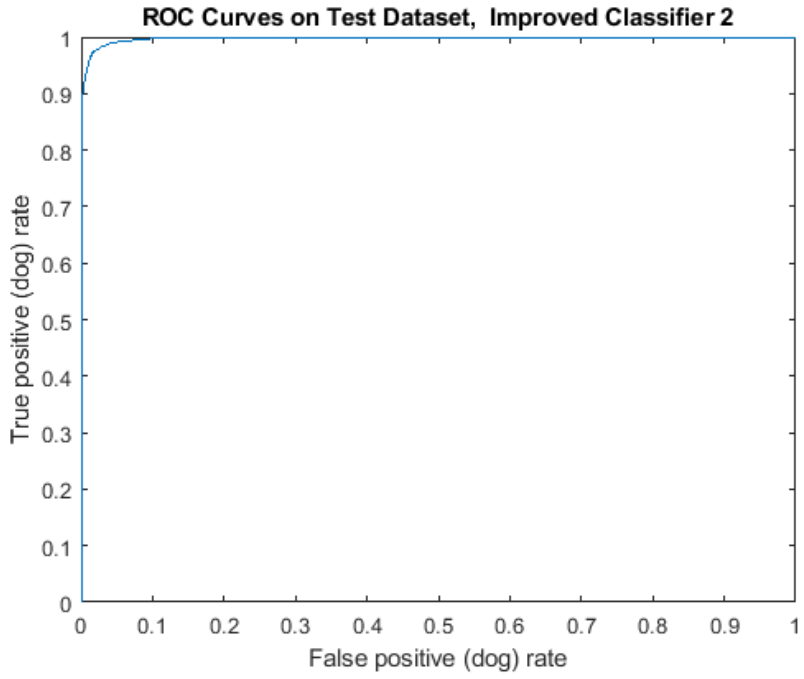


Exhibit 6 Samples of Correctly Classified Images – Improved Classifier 2

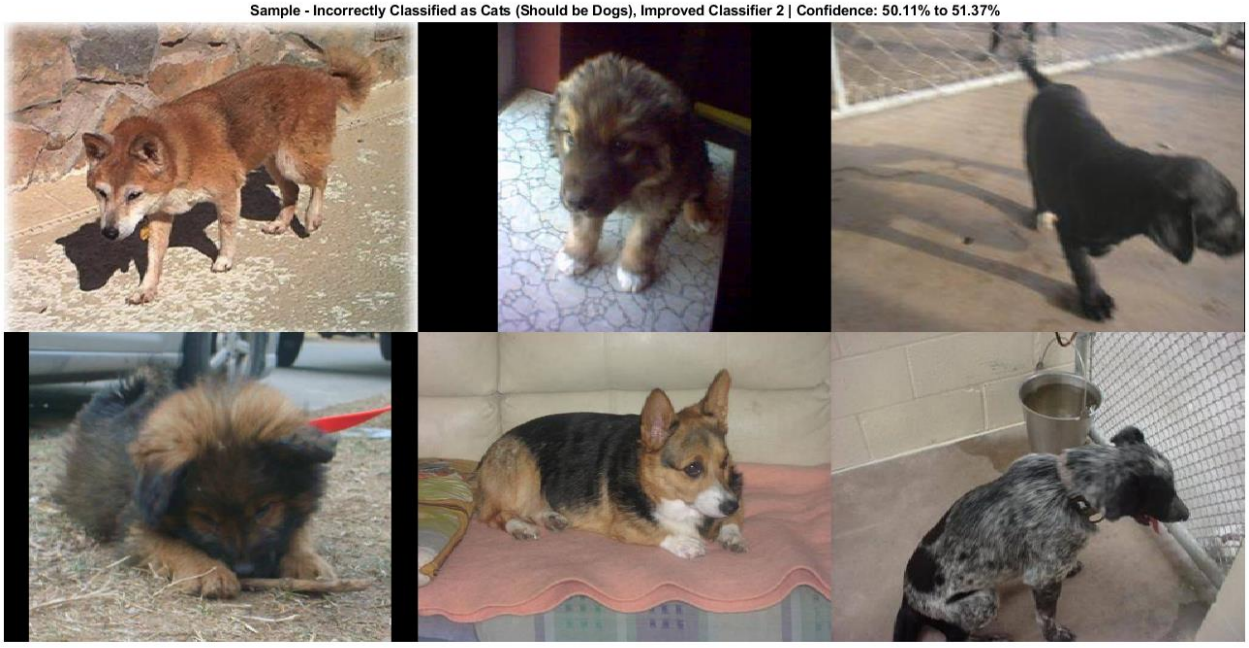
Sample - Correctly Classified as Cats, Improved Classifier 2 | Confidence: 100.00%

Sample - Correctly Classified as Dogs, Improved Classifier 2 | Confidence: 100.00%

Exhibit 7 Samples of Incorrectly Classified Images – Improved Classifier 2

Sample - Incorrectly Classified as Dogs (Should be Cats), Improved Classifier 2 | Confidence: 50.57% to 57.10%

Sample - Incorrectly Classified as Cats (Should be Dogs), Improved Classifier 2 | Confidence: 50.11% to 51.37%



Ensemble of Classifiers

The confusion matrix for the test dataset is shown in Figure 13, and the ROC curve is plotted and shown in Figure 14. The AUC is computed to be 0.9978. Samples of correctly (with the highest confidence) and incorrectly (with the lowest confidence) classified images are shown in Exhibit 8 and Exhibit 9 respectively.

Figure 13 Test Dataset Confusion Matrix – Ensemble of Classifiers

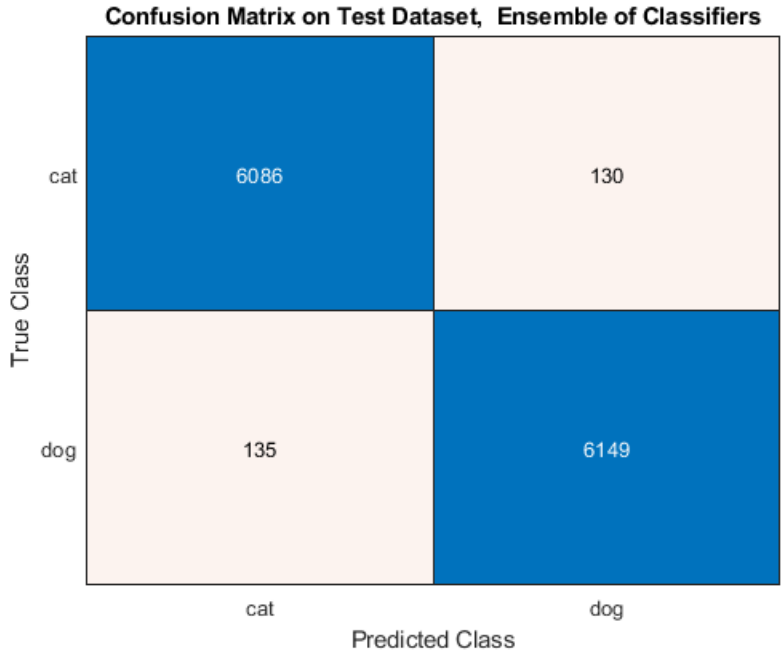


Figure 14 ROC Curve on Test Dataset – Ensemble of Classifiers

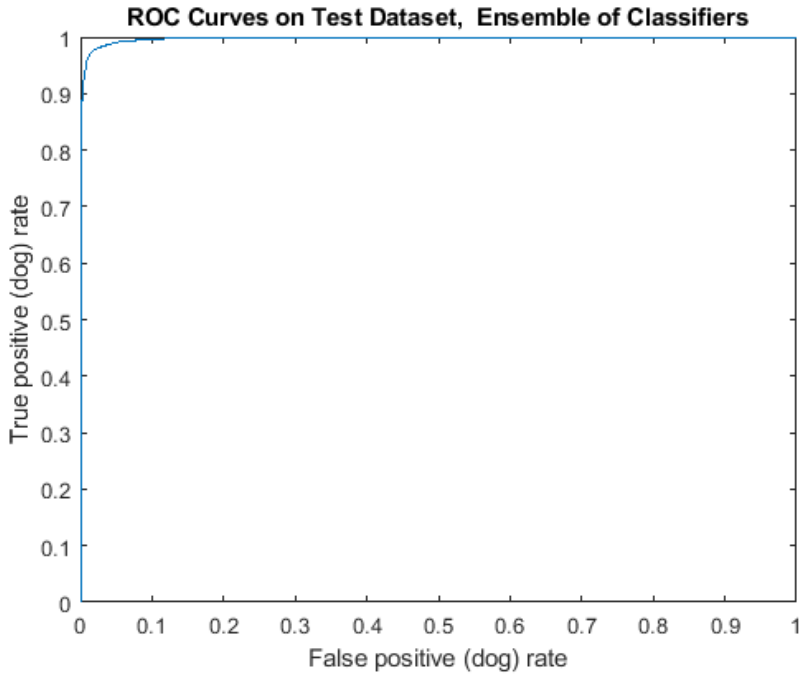
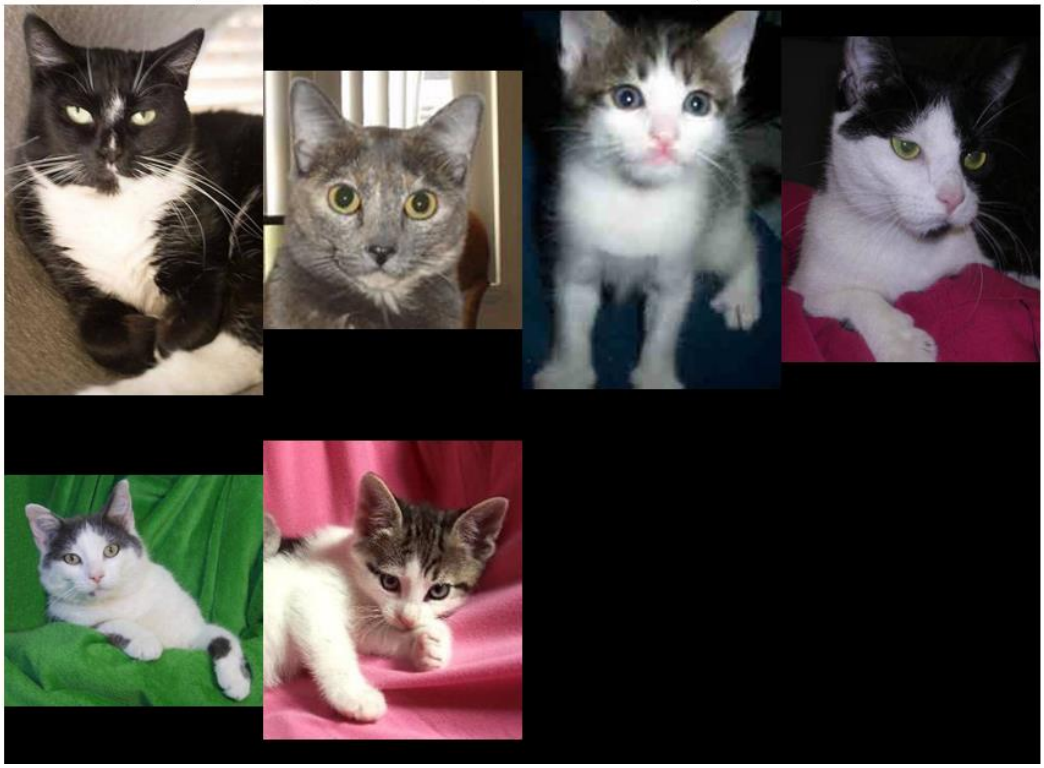


Exhibit 8 Samples of Correctly Classified Images – Ensemble of Classifiers

Sample - Correctly Classified as Cats, Ensemble of Classifiers | Confidence: 100.00%

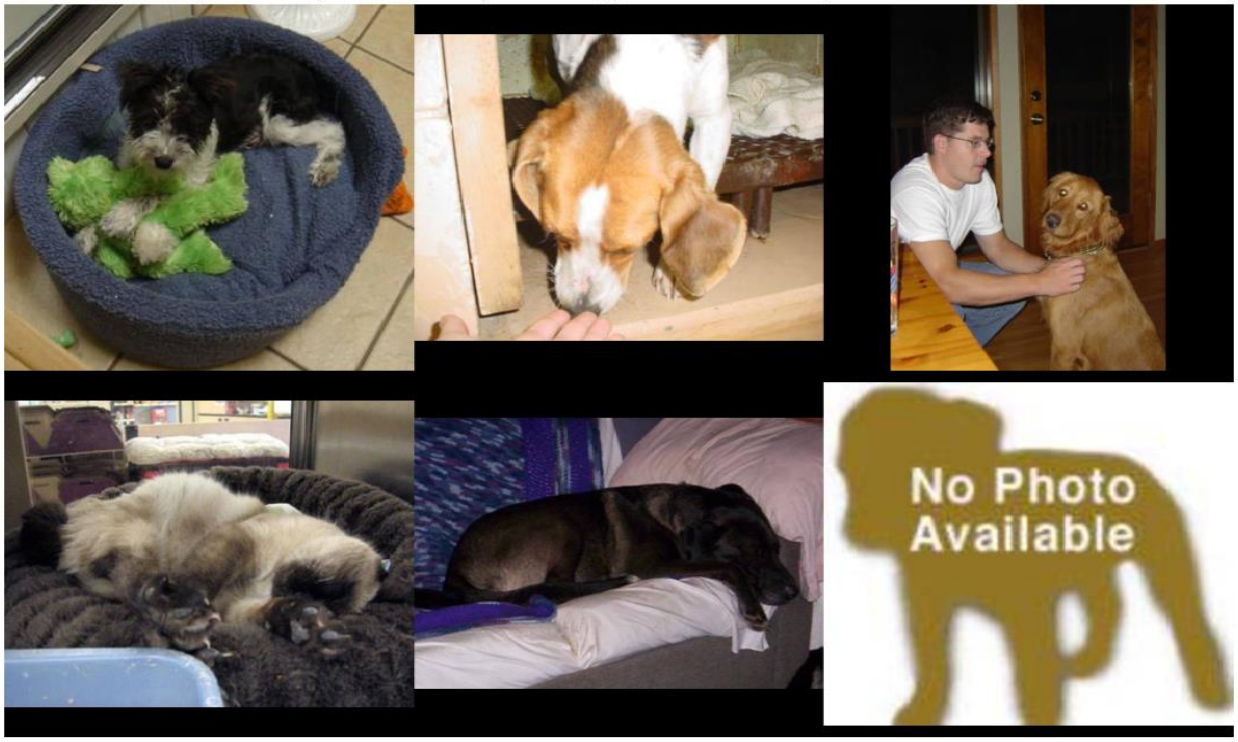


Sample - Correctly Classified as Dogs, Ensemble of Classifiers | Confidence: 100.00%

Exhibit 9 Samples of Incorrectly Classified Images – Ensemble of Classifiers

Sample - Incorrectly Classified as Dogs (Should be Cats), Ensemble of Classifiers | Confidence: 50.26% to 51.16%

Sample - Incorrectly Classified as Cats (Should be Dogs), Ensemble of Classifiers | Confidence: 50.18% to 52.27%



Questions

Below are the responses to the questions for this final project

1. What type of preprocessing is performed by the auxiliary function `readAndPreprocessImage`?

The auxiliary function `readAndPreprocessImage` re-sizes the images to 227 by 227 pixel which is an input format required by AlexNet. It also creates a 3-channel image if the image is grayscale, which is another requirement of AlexNet.

2. What can you say about the montage with network weights for the second convolutional layer (above)?

The montage with network weights represents the 96 feature maps used to capture various features of the input image in the first convolutional layer. These features include edges and circles of varying sizes, orientations, intensities and colors.

3. How many images are there in each set (training / validation)?

There are 28 images in the training set and 12 images in the validation set (in the warm-up exercise).

4. Is the validation accuracy (in my example 66.67%) acceptable for a two-class classifier? Why (not)? If not, what could be the problem?

No, it is not acceptable. The validation accuracy is 83.33% which is better than random guess but far from the accuracy achieved by humans. Humans could have easily and accurately classified those 40 images. The problem could be due to the lack of training samples as the model may not be trained to see images of cats and dogs in a wide variety of compositions. The problem could also be caused by overfitting as the training accuracy was increasing while the validation accuracy was decreasing.

5. Did your classifier recognize 'Doge' as a dog? If not, can you tell why?

No, my classifier did not recognize ‘Doge’ as a dog, and it classified ‘Doge’ as a cat with a very high confidence. It is likely due to the lack of image samples to train the model, and the ‘Doge’ resembles features of cats in this particular image especially its ears, eyes and nose.

6. Is the validation accuracy (in my example 75%) better than before? What could be the reason(s) behind such (modest) improvement? How could you improve it even further?

No, the accuracy decreased to 75.00%, and is worst than the case without data augmentation. This is likely due to a combination of reasons such as different hyperparameters and randomness factor caused by lack of training samples. The data augmentation scenario used a higher learning rate 0.0003 instead of 0.0001, and the model may not converge to a minimum due to larger step-size. Also, the data augmentation scenario did not shuffle the training samples in every epoch. Since the `MiniBatchSize` of 10 does not evenly divide the training sample of 28 images, the same 8 images would be discarded in every epoch, effectively reducing the sample size to 20 images in the data augmentation scenario. Shuffling images and using a lower learning rate may help improving the performance, but increasing the training samples would likely result in the greatest improvement.

7. Did the validation accuracy improve as a result of using a much larger training dataset? How could you improve it even further?

Yes, the validation accuracy improved to 96.02%. The model could be improved further by a variety of techniques, as outlined in the online e-book *"Neural Networks and Deep Learning"* by Michael Nielsen. Techniques to improve the performance include using regularization methods such as L1 and L1 regularizations and artificial expansion of the training data, and better choices of hyperparameters such as learning rate, mini batch size and number of training epochs. Data augmentation also helps as shown in the improved classifier 1.

Lessons Learned

The key lessons learned in this final project include:

1. While Dr. Marques provided a set of baseline hyperparameters in this final project, the choice of hyperparameters may not be trivial for new learners of neural networks if these baseline hyperparameters were not provided. As shown in the process of developing improved classifier 2, the learning rates between 0.0005 and 0.0001 could yield drastically different results. An incorrect choice of hyperparameters may result in the model perceived as "not working" or incorrectly configured. Debugging such a model may not be an easy task and could be frustrating for new learners as model training is very time-consuming. Using reduced dataset may help identify an initial set of "working" hyperparameters quickly.

2. Training a deep learning model is time-consuming. The three classifiers used in this final project were trained on two different machines: one machine with CPU only and the other machine with a GPU. While training on GPU reduced the training time by approximately 50% or more, the training time was still well over 3 hours. Although I have not had a chance to explore FAU's high performance computing (HPC) facility (https://hpc.fau.edu), it is available to FAU students, faculty and researchers and could be use to train deep learning models. Other computing options include Kaggle Kernels (https://www.kaggle.com/kernels) and Google Colab (https://colab.research.google.com). Both are cloud computing platforms provided by Google for machine learning programs, but they require programs to be written in Pythons/TensorFlow.

Conclusion

This final project shows that a small change in hyperparameters or training options could result in very different model performance. However, the greatest improvement could be achieved by increasing the training samples. Using ensemble of classifiers may further improve the classification accuracy, in the expense of time to train different models. The classification accuracy is also impacted by the quality of images as images with clear shots of cats and dogs were typically classified correctly with high confidence.

The deep learning workflow implementation in MATLAB is generally smooth. Pretrained neural networks such as AlexNet could be imported with just one command line. The implementation is made easier with a rich set of built-in functions in MATLAB for various deep learning models and tasks, as well as well-written MATLAB documentation and tutorials from MathWorks on machine learning, deep learning, and transfer learning.

References

[1] O. Marques, *CAP 4630 Introduction to Artificial Intelligence Final Project: Cats vs. Dogs Guidelines*, Florida Atlantic University, 2019.

[2] MathWorks, *MATLAB Documentation*, www.mathworks.com/help/matlab/index.html. Accessed July 21, 2019.

[3] M. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015, neuralnetworksanddeeplearning.com. Accessed July 21, 2019.

[4] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach, Third Edition*, Prentice Hall, 2010.

Appendix Page

Listing 1: MATLAB Published Output – Warm-up Exercise

CAP 4630 - Intro to AI - FAU - Dr. Marques - Summer 2019

Table of Contents

Student Name: Tsz Shing Tsoi

Final Project - Starter code (2-class classifier: cats vs. dogs)

```
% Inspired by the example "Deep Learning for Pet Classification"
% (Copyright 2016 The MathWorks, Inc.)
```

Part 1: Download, load and inspect Pre-trained Convolutional Neural Network (CNN)

```
% You will need to download a pre-trained CNN model for this example.
```

```
% There are several pre-trained networks that have gained popularity.

% Most of these have been trained on the ImageNet dataset, which has
 1000
% object categories and 1.2 million training images[1]. "AlexNet" is
 one
% such model [2].

% See
% https://www.mathworks.com/help/deeplearning/ug/pretrained-
convolutional-neural-networks.html
% for a list of pre-trained networks available in MATLAB.
```

1.1: Loading a pre-trained "AlexNet"

```
% Ensure that you have downloaded and installed the
% "Deep Learning Toolbox Model for AlexNet Network" support package.

% See https://www.mathworks.com/matlabcentral/fileexchange/59133-deep-
learning-toolbox-model-for-alexnet-network
% and
% https://www.mathworks.com/help/deeplearning/ug/pretrained-
convolutional-neural-networks.html
% for additional information.
clc;
clear;
close all;

model = alexnet;
```

1.3: Inspect the CNN's layers

```
model.Layers

% The intermediate layers make up the bulk of the CNN. These are a
 series
% of convolutional layers, interspersed with rectified linear units
 (ReLU)
% and max-pooling layers [2]. Following the these layers are 3
% fully-connected layers.

% The final layer is the classification layer and its properties
 depend on
% the classification task. In this example, the CNN model that was
 loaded
% was trained to solve a 1000-way classification problem. Thus the
% classification layer has 1000 classes from the ImageNet dataset.

% Inspect the last layer
model.Layers(end)

% Number of class names for ImageNet classification task
```

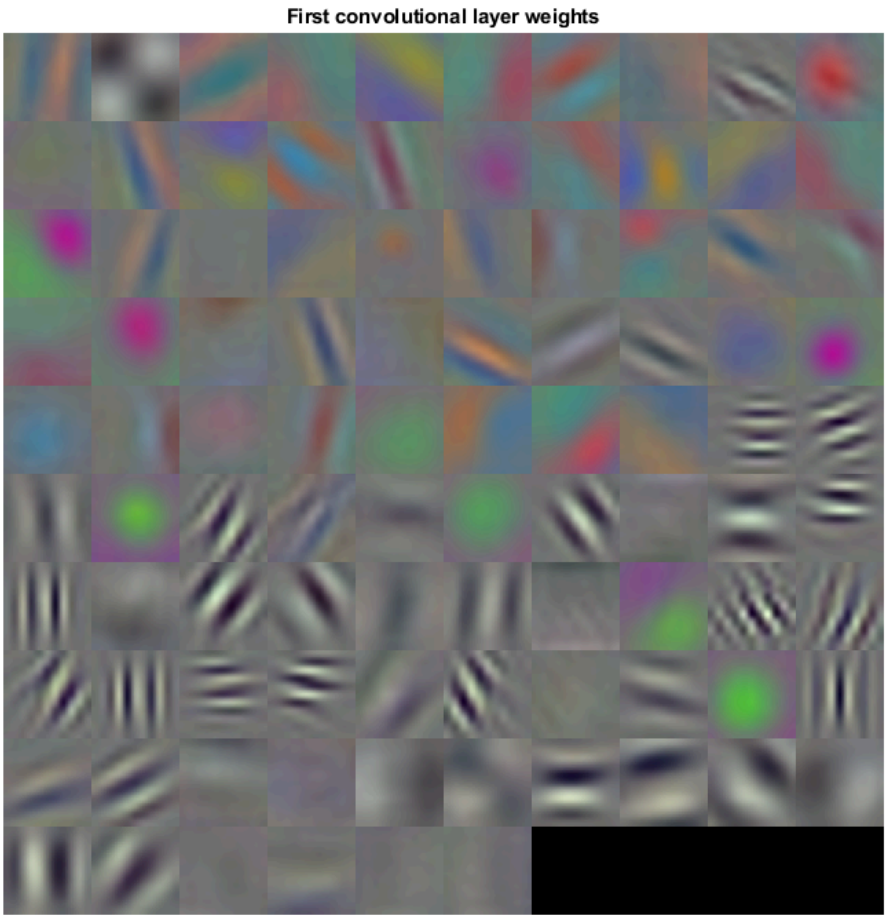
```
numel(model.Layers(end).ClassNames)

% Analyze the AlexNet in more detail using the network analyzer
% to display an interactive visualization of the network architecture
% and detailed information about the network layers.
analyzeNetwork(model)

% Note that the CNN model is not going to be used for the original
% classification task. It is going to be re-purposed to solve a
 different
% classification task on the pets dataset.


ans =

  25x1 Layer array with layers:

     1   'data'     Image Input                    227x227x3 images
 with 'zerocenter' normalization
     2   'conv1'    Convolution                    96 11x11x3
 convolutions with stride [4  4] and padding [0  0  0  0]
     3   'relu1'    ReLU                           ReLU
     4   'norm1'    Cross Channel Normalization    cross channel
 normalization with 5 channels per element
     5   'pool1'    Max Pooling                    3x3 max pooling with
 stride [2  2] and padding [0  0  0  0]
     6   'conv2'    Grouped Convolution            2 groups of 128
 5x5x48 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'relu2'    ReLU                           ReLU
     8   'norm2'    Cross Channel Normalization    cross channel
 normalization with 5 channels per element
     9   'pool2'    Max Pooling                    3x3 max pooling with
 stride [2  2] and padding [0  0  0  0]
    10   'conv3'    Convolution                    384 3x3x256
 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'relu3'    ReLU                           ReLU
    12   'conv4'    Grouped Convolution            2 groups of 192
 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    13   'relu4'    ReLU                           ReLU
    14   'conv5'    Grouped Convolution            2 groups of 128
 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    15   'relu5'    ReLU                           ReLU
    16   'pool5'    Max Pooling                    3x3 max pooling with
 stride [2  2] and padding [0  0  0  0]
    17   'fc6'      Fully Connected                4096 fully connected
 layer
    18   'relu6'    ReLU                           ReLU
    19   'drop6'    Dropout                        50% dropout
    20   'fc7'      Fully Connected                4096 fully connected
 layer
    21   'relu7'    ReLU                           ReLU
    22   'drop7'    Dropout                        50% dropout
    23   'fc8'      Fully Connected                1000 fully connected
 layer
```

```
    24   'prob'     Softmax                       softmax
    25   'output'   Classification Output         crossentropyex with
 'tench' and 999 other classes

ans =

  ClassificationOutputLayer with properties:

           Name: 'output'
        Classes: [1000×1 categorical]
     OutputSize: 1000

   Hyperparameters
    LossFunction: 'crossentropyex'


ans =

        1000
```

1.4: Inspect the network weights for the second convolutional layer

```
% Each layer of a CNN produces a response, or activation, to an input
% image. However, there are only a few layers within a CNN that are
% suitable for image feature extraction. The layers at the beginning
 of the
% network capture basic image features, such as edges and blobs. To
 see
% this, visualize the network filter weights from the first
 convolutional
% layer. This can help build up an intuition as to why the features
% extracted from CNNs work so well for image recognition tasks. Note
 that
% visualizing deeper layer weights is beyond the scope of this
 example. You
% can read more about that in the work of Zeiler and Fergus [4].

% Get the network weights for the second convolutional layer
w1 = model.Layers(2).Weights;

% Scale and resize the weights for visualization
w1 = mat2gray(w1);
w1 = imresize(w1,5);

% Display a montage of network weights.
figure
montage(w1)
title('First convolutional layer weights')

% Notice how the first layer of the network has learned filters for
```

```
% capturing blob and edge features. These "primitive" features are
 then
% processed by deeper network layers, which combine the early features
 to
% form higher level image features. These higher level features are
 better
% suited for recognition tasks because they combine all the primitive
% features into a richer image representation [5].
```



Part 2: Set up image data

2.1: Load simplified dataset and build image store

```
rng(0,'twister');

dataFolder = './data/PetImages';
categories = {'cat', 'dog'};
```

```
imds = imageDatastore(fullfile(dataFolder,
 categories), 'LabelSource', 'foldernames');
tbl = countEachLabel(imds);
disp (tbl)

% Use the smallest overlap set
% (useful when the two classes have different number of elements but
 not
% needed in this case)
minSetCount = min(tbl{:,2});

% Use splitEachLabel method to trim the set.
imds = splitEachLabel(imds, minSetCount, 'randomize');

% Notice that each set now has exactly the same number of images.
countEachLabel(imds)
```

```
    Label    Count
    _____    _____

     cat      20
     dog      20


ans =

  2×2 table

    Label    Count
    _____    _____

     cat      20
     dog      20
```

2.2: Pre-process Images For CNN

AlexNet can only process RGB images that are 227-by-227. To avoid re-saving all the images to this format, setup the `imds` read function, `imds.ReadFcn`, to pre-process images on-the-fly. The `imds.ReadFcn` is called every time an image is read from the `ImageDatastore`.

Set the ImageDatastore ReadFcn

```
imds.ReadFcn = @(filename)readAndPreprocessImage(filename);
```

2.3: Divide data into training and validation sets

```
[trainingSet, validationSet] = splitEachLabel(imds,
 0.7, 'randomized');

countEachLabel(trainingSet)
countEachLabel(validationSet)
```

```
ans =

  2×2 table

    Label    Count
    _____    _____

     cat      14
     dog      14


ans =

  2×2 table

    Label    Count
    _____    _____

     cat       6
     dog       6
```

Part 3: Transfer Learning

```
% The convolutional layers of the network extract image features that
 the
% last learnable layer and the final classification layer use to
 classify
% the input image.

% To retrain a pretrained network to classify new images, we must
 replace these
% last layers with new layers adapted to the new data set.
```

3.1: Freeze all but last three layers

```
layersTransfer = model.Layers(1:end-3);
numClasses = 2; % cat and dog

layers = [
    layersTransfer

 fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',2
    softmaxLayer
    classificationLayer];
```

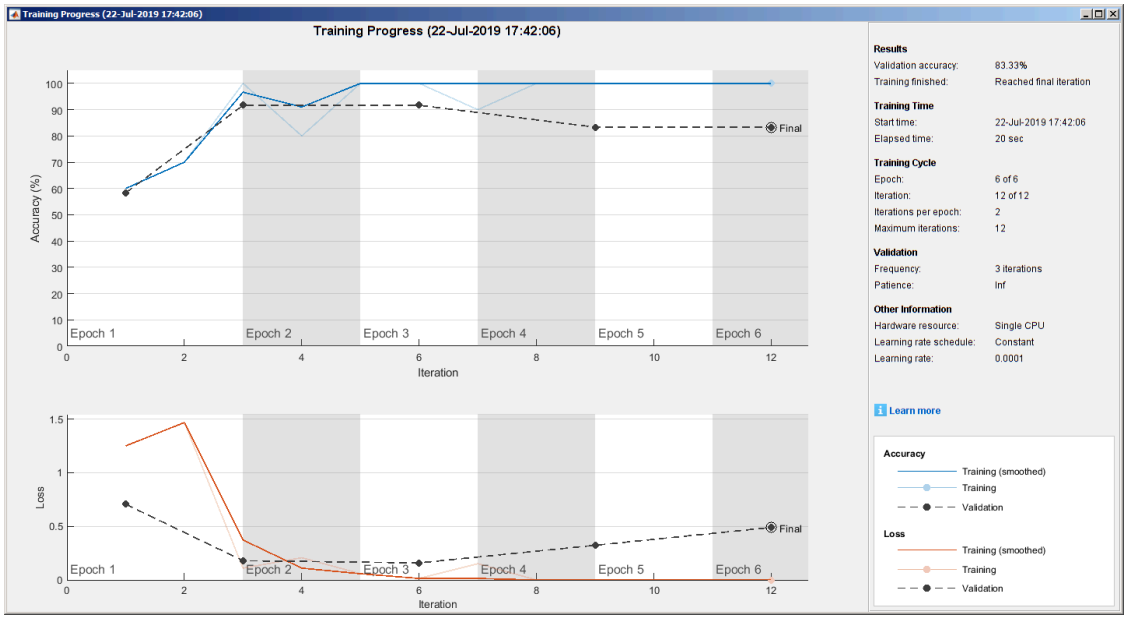
3.2: Configure training options

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',10, ...
    'MaxEpochs',6, ...
```

```
'InitialLearnRate',1e-4, ...
'Shuffle','every-epoch', ...
'ValidationData',validationSet, ...
'ValidationFrequency',3, ...
'Verbose',false, ...
'Plots','training-progress');
```

3.3: Retrain network

```
modelTransfer = trainNetwork(trainingSet,layers,options);
```



3.4: Classify the validation images using the fine-tuned network.

```
[YPred,scores] = classify(modelTransfer,validationSet);
```

3.5: Calculate the classification accuracy on the validation set.

Accuracy is the fraction of labels that the network predicts correctly.

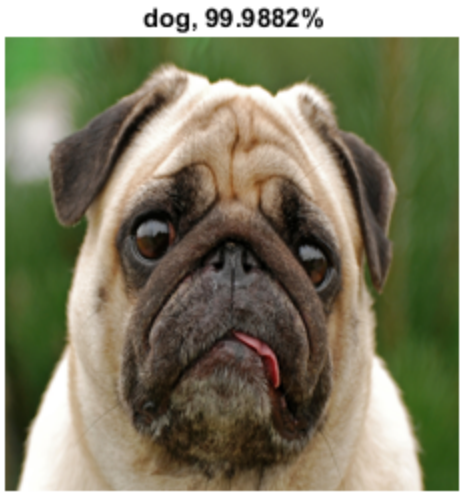
```
YValidation = validationSet.Labels;
accuracy = mean(YPred == YValidation);
fprintf("The validation accuracy is: %.2f %%\n", accuracy * 100);
```

The validation accuracy is: 83.33 %

3.6: Test it on unseen images

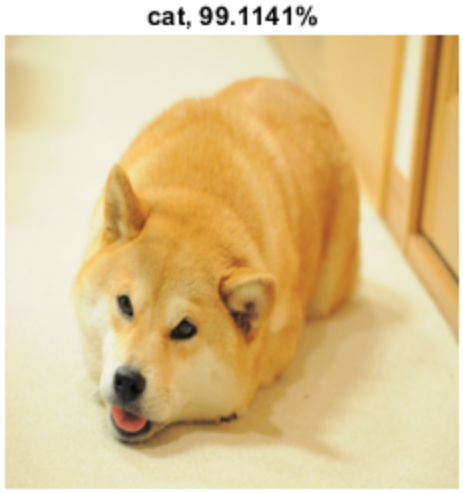
```
newImage1 = './dog.jpg'; % any dog image should do!
```

```
img1 = readAndPreprocessImage(newImage1);
YPred1 = predict(modelTransfer,img1);
[confidence1,idx1] = max(YPred1);
label1 = categories{idx1};
% Display test image and assigned label
figure
imshow(img1)
title(string(label1) + ", " + num2str(100*confidence1) + "%");

newImage2 = './cat.jpg'; % any cat image should do!
img2 = readAndPreprocessImage(newImage2);
YPred2 = predict(modelTransfer,img2);
[confidence2,idx2] = max(YPred2);
label2 = categories{idx2};
% Display test image and assigned label
figure
imshow(img2)
title(string(label2) + ", " + num2str(100*confidence2) + "%");
```



3.7: Test it on unseen images: Your turn!

```
% What about the iconic "Doge"?
% ENTER YOUR CODE HERE
newImage3 = './doge.jpg'; % any dog image should do!
img3 = readAndPreprocessImage(newImage3);
YPred3 = predict(modelTransfer,img3);
[confidence3,idx3] = max(YPred3);
label3 = categories{idx3};
% Display test image and assigned label
figure
imshow(img3)
title(string(label3) + ", " + num2str(100*confidence3) + "%");
```



Part 4: Data augmentation

```
% Data augmentation helps prevent the network from overfitting and
% memorizing the exact details of the training images.

% In MATLAB, this can be done using the "Augmented Image Datastore"
% (https://www.mathworks.com/help/deeplearning/ref/
augmentedimagedatastore.html)

% Despite its name, however, it DOES NOT increase the actual number of
% samples. When you use an augmented image datastore as a source of
% training images, the datastore randomly perturbs the training data
 for
% each epoch, so that each epoch uses a slightly different data set.
 The
% actual number of training images at each epoch does not change. The
% transformed images are not stored in memory.
```

4.1: Defining the imageAugmenter object

In our case, we shall use an augmented image datastore to randomly flip the training images along the vertical axis and randomly translate them up to 30 pixels and scale them up to 10% horizontally and vertically.

```
pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange);
```

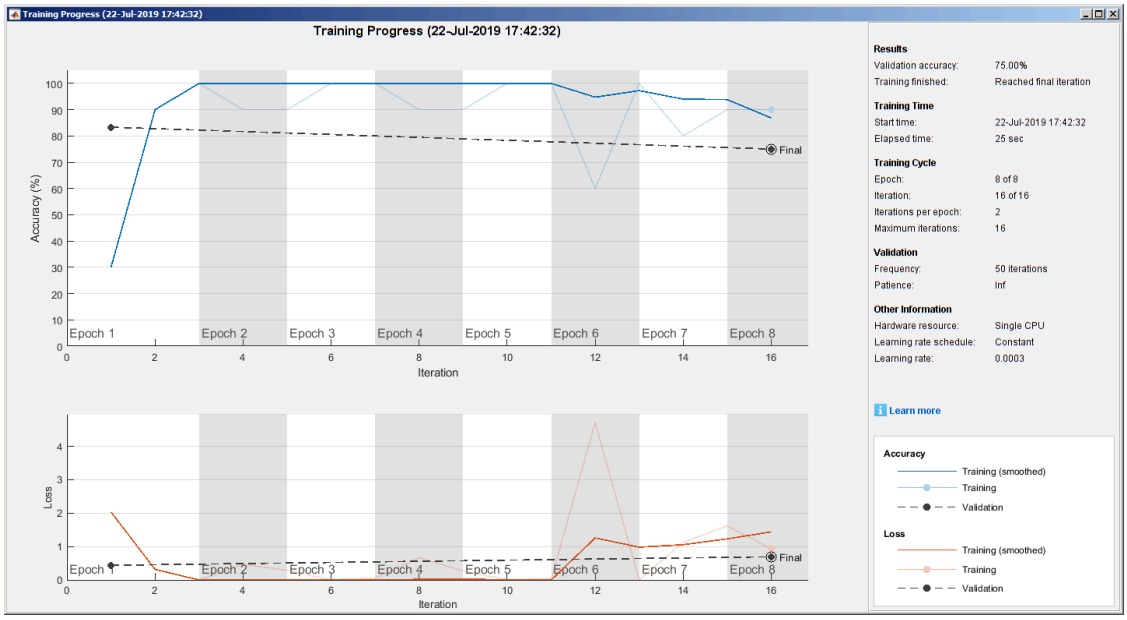
4.2: Building the augmented training and validation sets

```
inputSize = model.Layers(1).InputSize;
augimdsTrain = augmentedImageDatastore(inputSize(1:2),trainingSet, ...
    'DataAugmentation',imageAugmenter);

disp(augimdsTrain.NumObservations) % You should see 28

augimdsValidation =
 augmentedImageDatastore(inputSize(1:2),validationSet);

disp(augimdsValidation.NumObservations) % You should see 12
```

```
    28

    12
```

4.3: Train the network with augmented datasets

```
miniBatchSize = 10;
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',8, ...
    'InitialLearnRate',3e-4, ...
    'ValidationData',augimdsValidation, ...
    'Verbose',false, ...
    'Plots','training-progress');

modelAug = trainNetwork(augimdsTrain,layers,options);
```



4.4: Classify the validation images using the fine-tuned network.

```
[YPredAug,probsAug] = classify(modelAug,augimdsValidation); %changed
 from augimdsValidation to validationSet
```

4.5: Calculate the classification accuracy on the validation set.

Accuracy is the fraction of labels that the network predicts correctly.

```
YValidationAug = validationSet.Labels;
accuracyAug = mean(YPredAug == YValidationAug);
fprintf("The validation accuracy is: %.2f %%\n", accuracyAug * 100);
```

```
The validation accuracy is: 75.00 %
```

Part 5: Larger datasets

```
% One possible reason why the accuracy of the classifier was so low
% might have to do with not enough training data.
% Of course, there are plenty of dogs and cats around (in the Kaggle
% dataset and elsewhere) to circumvent this problem.

% In this part, you will use the Kaggle dataset and basically repeat
 the
% steps in Parts 2 and 3 (and 4, if you wish) using larger training
 and
% validation datasets.

% See Guidelines for instructions.
```

5.1: Set up image data from full Kaggle training dataset

```
rng(0,'twister');

% Load full training dataset and build image store
dataFolderFull = './data/train';
catFilesFull = fullfile(dataFolderFull,'cat.*.jpg');
dogFilesFull = fullfile(dataFolderFull,'dog.*.jpg');
dataLabelsFull =
 [repmat(categorical({'cat'}),numel(dir(catFilesFull)),1);...
    repmat(categorical({'dog'}),numel(dir(dogFilesFull)),1)];
imdsFull = imageDatastore({catFilesFull dogFilesFull},'Labels',
 dataLabelsFull);
tblFull = countEachLabel(imdsFull);
disp (tblFull)
```

```
% Use the smallest overlap set
% (useful when the two classes have different number of elements)
minSetCountFull = min(tblFull{:,2});

% Use splitEachLabel method to trim the set.
imdsFull = splitEachLabel(imdsFull, minSetCountFull, 'randomize');

% Notice that each set now has exactly the same number of images.
countEachLabel(imdsFull)

% Set the ImageDatastore ReadFcn
imdsFull.ReadFcn = @(filename)readAndPreprocessImage(filename);

% Divide data into training and validation sets
[trainingSetFull, validationSetFull] = splitEachLabel(imdsFull,
 0.8, 'randomized');

countEachLabel(trainingSetFull)
countEachLabel(validationSetFull)
```

```
    Label    Count
    _____    _____

     cat     12500
     dog     12500


ans =

  2×2 table

    Label    Count
    _____    _____

     cat     12500
     dog     12500


ans =

  2×2 table

    Label    Count
    _____    _____

     cat     10000
     dog     10000


ans =

  2×2 table
```

```
    Label    Count
    _____    _____

     cat     2500
     dog     2500
```

5.2: Transfer learning

```
fileNameTemp = fullfile('./modelTransferFull.mat');

% If trained model file does not exists, train model; otherwise load
 model
if ~exist(fileNameTemp,'file')

    % create checkpoint folder if not exist
    if ~exist(fullfile('./checkpoint'),'dir')
        mkdir(fullfile('./checkpoint'))
    end

    % Configure training options
    optionsFull = trainingOptions('sgdm', ...
        'MiniBatchSize',10, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',1e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',validationSetFull, ...
        'ValidationFrequency',100, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'CheckPointPath',fullfile('./checkpoint'));

    % Retrain network
    modelTransferFull =
 trainNetwork(trainingSetFull,layers,optionsFull);
    save(fileNameTemp,'modelTransferFull');
else
    load(fileNameTemp,'modelTransferFull');
end
```

5.3: Classify the validation images using the fine-tuned network.

```
[YPredFull,scoresFull] =
 classify(modelTransferFull,validationSetFull);

% Calculate the classification accuracy on the validation set.
% Accuracy is the fraction of labels that the network predicts
 correctly.
YValidationFull = validationSetFull.Labels;
accuracyFull = mean(YPredFull == YValidationFull);
```

```
fprintf("The validation accuracy is: %.2f %%\n", accuracyFull * 100);
```

```
The validation accuracy is: 96.02 %
```

References

[1] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3] Vedaldi, Andrea, and Karel Lenc. "MatConvNet-convolutional neural networks for MATLAB." arXiv preprint arXiv:1412.4564 (2014).

[4] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." Computer Vision-ECCV 2014. Springer International Publishing, 2014. 818-833.

[5] Donahue, Jeff, et al. "Decaf: A deep convolutional activation feature for generic visual recognition." arXiv preprint arXiv:1310.1531 (2013).

Published with MATLAB® R2019a

Listing 2: MATLAB Published Output – Final Solution

CAP 4630 - Intro to AI - FAU - Dr. Marques - Summer 2019

Table of Contents

Student Name: Tsz Shing Tsoi

Final Project - Final Solution (2-class classifier: cats vs. dogs)

Part 1: Data Setup

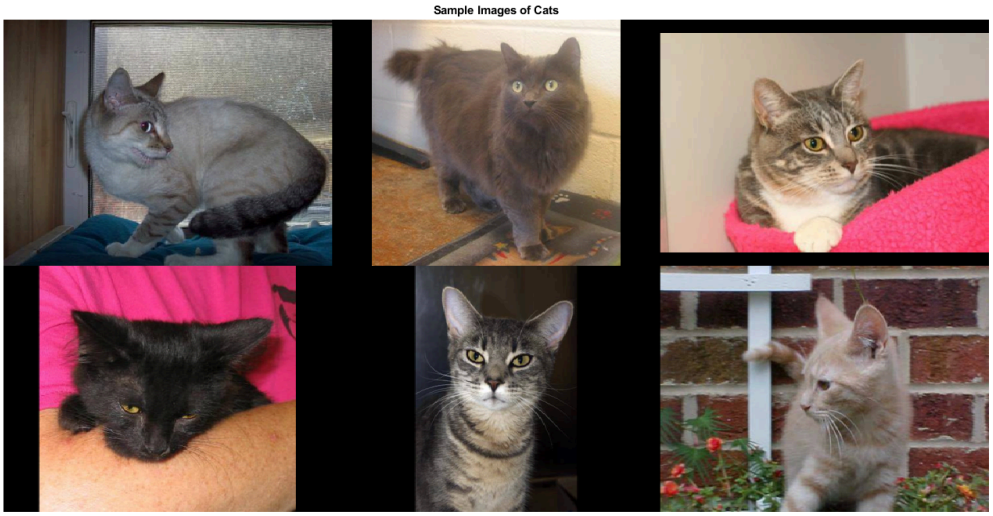
In this part, the full Kaggle training dataset will be loaded and partitioned into training and hold-out validation sets. These training and validation datasets will be used for developing baseline classifier, improved classifier 1, and improved classifier 2.

1.1: Load image data from full Kaggle training dataset

```
clc;
clear;
close all;

% Load full training dataset and build image store
dataFolderFull = './data/train';
catFilesFull = fullfile(dataFolderFull,'cat.*.jpg');
dogFilesFull = fullfile(dataFolderFull,'dog.*.jpg');
dataLabelsFull =
 [repmat(categorical({'cat'}),numel(dir(catFilesFull)),1);...
    repmat(categorical({'dog'}),numel(dir(dogFilesFull)),1)];
imdsFull = imageDatastore({catFilesFull dogFilesFull},'Labels',
 dataLabelsFull);
```

1.2: Display sample images

```
rng(0,'twister'); % seed random generator for consistency

% extract indices of cats and dogs
idxCat = find(imdsFull.Labels == 'cat');
idxDog = find(imdsFull.Labels == 'dog');

% randomly select up to 6 images
selectIdxCat= randperm(size(idxCat,1),min(size(idxCat,1),6));
selectIdxDog= randperm(size(idxDog,1),min(size(idxCat,1),6));

% diaplay montage of images
figure, montage(subset(imdsFull, idxCat(selectIdxCat)));
title('Sample Images of Cats');
figure, montage(subset(imdsFull, idxDog(selectIdxDog)));
title('Sample Images of Dogs');
```



1.3: Set up image data store

```
rng(0,'twister'); % seed random generator for consistency

tblFull = countEachLabel(imdsFull);
disp (tblFull)

% Use the smallest overlap set
% (useful when the two classes have different number of elements)
minSetCountFull = min(tblFull{:,2});

% Use splitEachLabel method to trim the set.
imdsFull = splitEachLabel(imdsFull, minSetCountFull, 'randomize');

% Set the ImageDatastore ReadFcn
```

```
imdsFull.ReadFcn = @(filename)readAndPreprocessImage(filename);

% Divide data into training and validation sets
[trainingSetFull, validationSetFull] = splitEachLabel(imdsFull,
 0.8, 'randomized');

countEachLabel(trainingSetFull)
countEachLabel(validationSetFull)
```

```
    Label    Count
    _____    _____

     cat     12500
     dog     12500


ans =

  2×2 table

    Label    Count
    _____    _____

     cat     10000
     dog     10000


ans =

  2×2 table

    Label    Count
    _____    _____

     cat     2500
     dog     2500
```

1.4: Set up layers for the modified CNN

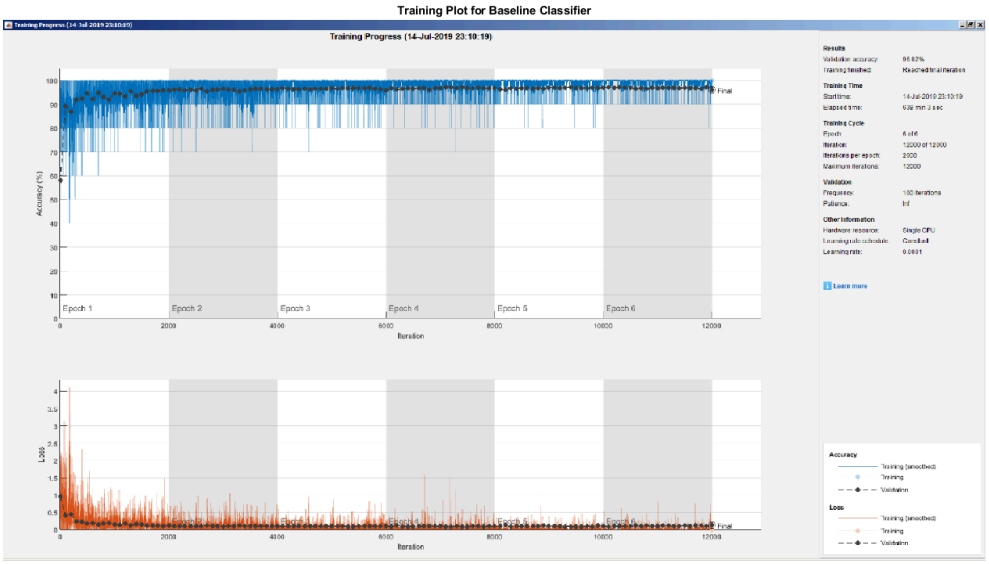
```
model = alexnet;

% Freeze all but last three layers
layersTransferFull = model.Layers(1:end-3);
numClasses = 2; % cat and dog

layersFull = [
    layersTransferFull

 fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',2
    softmaxLayer
    classificationLayer];
```

```
% create checkpoint folder if not exist
if ~exist(fullfile('./checkpoint'),'dir')
    mkdir(fullfile('./checkpoint'))
end
```

Part 2: Baseline Classifier

2.1: Transfer Learning for Baseline Classifier

```
fileNameTemp = fullfile('./modelTransferFull.mat');

% If trained model file does not exists, train model; otherwise load
 model
if ~exist(fileNameTemp,'file')

    % Configure training options
    % ValidationFrequency does not affect the final results and value
 could be
    % increased to decrease the training time.
    optionsFull = trainingOptions('sgdm', ...
        'MiniBatchSize',10, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',1e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',validationSetFull, ...
        'ValidationFrequency',100, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'CheckPointPath',fullfile('./checkpoint'), ...
        'OutputFcn',
 @(info)saveTrainingPlot(info, 'modelTransferFullTrainingPlot.png'));

    % close all figures if opened
    close(findall(groot, 'Type', 'Figure'));
    % Retrain network
    modelTransferFull =
 trainNetwork(trainingSetFull,layersFull,optionsFull);
    save(fileNameTemp,'modelTransferFull');
else
    load(fileNameTemp,'modelTransferFull');
    figure, imshow(imread(fullfile('./
modelTransferFullTrainingPlot.png')));
    title('Training Plot for Baseline Classifier');
end
```



2.2: Classify the validation images using the fine-tuned network.

```
fileNameTemp = fullfile('./YPredFull.mat');
if ~exist(fileNameTemp,'file')
    [YPredFull,scoresFull] =
 classify(modelTransferFull,validationSetFull);
    save(fileNameTemp,'YPredFull');
else
    load(fileNameTemp,'YPredFull');
end

% Calculate the classification accuracy on the validation set.
% Accuracy is the fraction of labels that the network predicts
 correctly.
YValidationFull = validationSetFull.Labels;
accuracyFull = mean(YPredFull == YValidationFull);
fprintf("The validation accuracy for Baseline Classifier is: %.2f %%
\n", accuracyFull * 100);
```

```
The validation accuracy for Baseline Classifier is: 96.02 %
```

Part 3: Improved Classifier 1

Improved Classifier 1 uses data augmentation. Data augmentation helps prevent the network from over-fitting and memorizing the exact details of the training images.

3.1: Defining the imageAugmenter object

In our case, we shall use an augmented image datastore to randomly flip the training images along the vertical axis and randomly translate them up to 30 pixels and scale them up to 10% horizontally and vertically.

```
pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange);
```

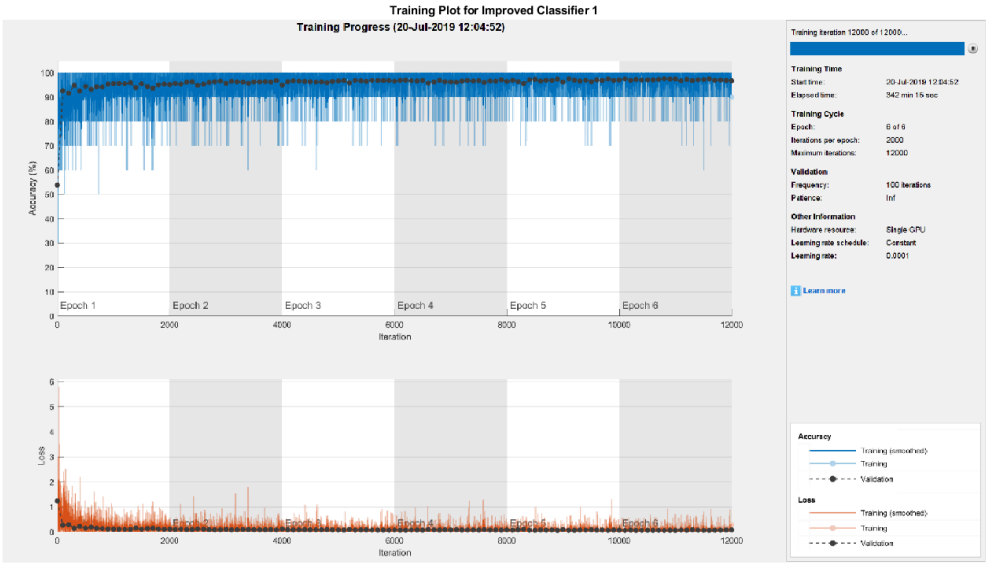
3.2: Building the augmented training and validation sets

```
inputSize = model.Layers(1).InputSize;
augimdsTrainFull =
 augmentedImageDatastore(inputSize(1:2),trainingSetFull, ...
    'DataAugmentation',imageAugmenter);
augimdsValidationFull =
 augmentedImageDatastore(inputSize(1:2),validationSetFull);
```

3.3: Train the network with augmented datasets

```
fileNameTemp = fullfile('./modelAugFull.mat');

% If trained model file does not exists, train model; otherwise load
 model
if ~exist(fileNameTemp,'file')

    % Configure training options
    % Same as the baseline classifier
    optionsAugFull = trainingOptions('sgdm', ...
        'MiniBatchSize',10, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',1e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',augimdsValidationFull, ...
        'ValidationFrequency',100, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'CheckPointPath',fullfile('./checkpoint'),...
        'OutputFcn',
 @(info)saveTrainingPlot(info, 'modelAugFullTrainingPlot.png'));

    % close all figures if opened
    close(findall(groot, 'Type', 'Figure'));
```

```
    % Retrain network
    modelAugFull =
 trainNetwork(augimdsTrainFull,layersFull,optionsAugFull);
    save(fileNameTemp,'modelAugFull');
else
    load(fileNameTemp,'modelAugFull');
    figure, imshow(imread(fullfile('./
modelAugFullTrainingPlot.png')));
    title('Training Plot for Improved Classifier 1');
end
```



3.4: Classify the validation images using the fine-tuned network.

```
fileNameTemp = fullfile('./YPredAugFull.mat');
if ~exist(fileNameTemp,'file')
    [YPredAugFull,probsAugFull] =
 classify(modelAugFull,augimdsValidationFull);
    save(fileNameTemp,'YPredAugFull');
else
    load(fileNameTemp,'YPredAugFull');
end

% Calculate the classification accuracy on the validation set.
% Accuracy is the fraction of labels that the network predicts
 correctly.
YValidationAugFull = validationSetFull.Labels;
accuracyAugFull = mean(YPredAugFull == YValidationAugFull);
fprintf("The validation accuracy for Improved Classifier 1 is: %.2f %%
\n", accuracyAugFull * 100);
```

```
The validation accuracy for Improved Classifier 1 is: 96.72 %
```

Part 4: Improved Classifier 2

This part attempts to identify the optimal set of hyperparameters, including learning rate, mini-batch size and number of epochs using heuristics. The heuristics used is a reduced dataset, i.e. 5% of the full training dataset. Different learning rates and mini-batch sizes are tested to identify the optimal values. Variable learning rates may be used to speed up training if deemed appropriate. The number of epoches is then determined based on the optimal set of learning rate and mini-batch size. The Improved Classifier 2 are then developed based on the optimal set of hyperparameters.

4.1: Split training and validation sets as heuristics

```
%seed random generator for consistency
rng(0,'twister');
trainingSetHeu = splitEachLabel(trainingSetFull, 0.05, 'randomized');
validationSetHeu = splitEachLabel(validationSetFull,
 0.05, 'randomized');
```

4.2: Set the set of hyperparaters to be tested

```
% Test 3 different learning rates.
% The first element is control.
testLearnRate = [1e-4, ...
    1e-3, 5e-4, 1e-5, ...
    1e-4, 1e-4, 1e-4]';

% Test 3 different mini-batch sizes
testMiniBatchSize = [10, ...
    10, 10, 10,  ...
    100, 50, 5]';

% Placeholder for final validation accuracy
testValidationAcc = zeros(size(testLearnRate,1), 1);

testTable = table(testLearnRate, testMiniBatchSize,
 testValidationAcc, ...
    'VariableNames',
 {'LearningRate' 'MiniBatchSize' 'ValidationAccuracy'});
```

4.3: Train model with different hyperparameters on heuristics dataset

```
fileNameTemp = fullfile('./modelTestHeuInfoSummary.mat');

% If the result matrix file does not exists, conduct test; otherwise
 load
% the result matrix file
if ~exist(fileNameTemp,'file')
```

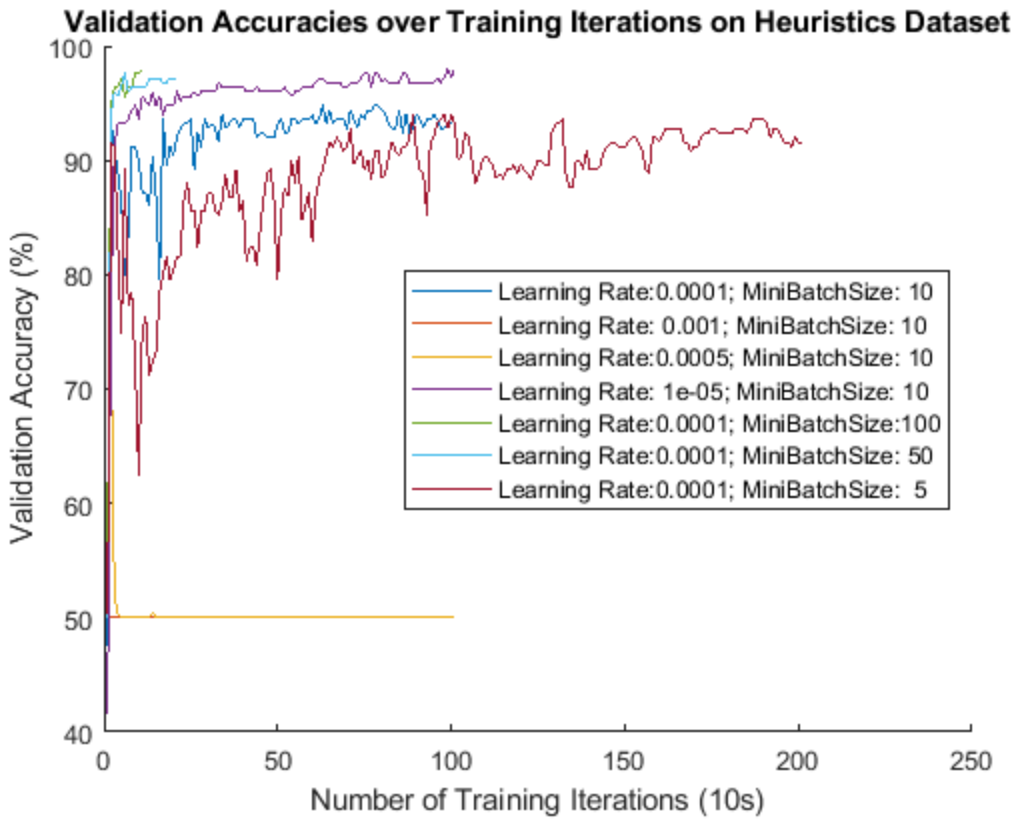
```
    modelTestHeuInfoSummary = struct;
    for i = 1:size(testTable, 1)

         %seed random generator for consistency
         rng(0,'twister');

        % Configure training options
        % MaxEpochs is increased to account for decreased learning
 rate in some
        % scenarios
        optionsHeu = trainingOptions('sgdm', ...
            'MiniBatchSize',testTable.MiniBatchSize(i), ...
            'MaxEpochs',10, ...
            'InitialLearnRate',testTable.LearningRate(i), ...
            'Shuffle','every-epoch', ...
            'ValidationData',validationSetHeu, ...
            'ValidationFrequency',10, ...
            'Verbose',false);

        % train network
        fprintf("Training scenario %d begins...\n", i);
        [modelTestHeu, modelTestHeuInfo] =
 trainNetwork(trainingSetHeu,layersFull,optionsHeu);
        fprintf("Training scenario %d ended.\n", i);

        % save network info
        testTable.ValidationAccuracy(i) =
 modelTestHeuInfo.ValidationAccuracy(end);
        modelTestHeuInfoSummary(i).modelInfo = modelTestHeuInfo;
        save(fileNameTemp, 'modelTestHeuInfoSummary');
    end
else
    load(fileNameTemp,'modelTestHeuInfoSummary');
    for i = 1:min(size(testTable, 1),size(modelTestHeuInfoSummary, 2))
        testTable.ValidationAccuracy(i) =
 modelTestHeuInfoSummary(i).modelInfo.ValidationAccuracy(end);
    end
end
```

4.4: Display final validation accuracy and plot training accuracy for each scenario

```
testTable

figure, hold on
for i =1:size(modelTestHeuInfoSummary, 2)
    plot
 (rmmissing(modelTestHeuInfoSummary(i).modelInfo.ValidationAccuracy))
end
legend(strcat('Learning Rate: ',num2str(testTable.LearningRate), ...
    '; MiniBatchSize: ',int2str(testTable.MiniBatchSize)), ...
    'location','east');
```

```
title('Validation Accuracies over Training Iterations on Heuristics
 Dataset');
xlabel('Number of Training Iterations (10s)');
ylabel('Validation Accuracy (%)');
hold off
```

```
testTable =

  7×3 table

    LearningRate    MiniBatchSize    ValidationAccuracy
    ____________    _____________    __________________

       0.0001            10               93.6
        0.001            10               50
       0.0005            10               50
        1e-05            10               98
       0.0001           100               98
       0.0001            50               97.2
       0.0001             5               91.6
```

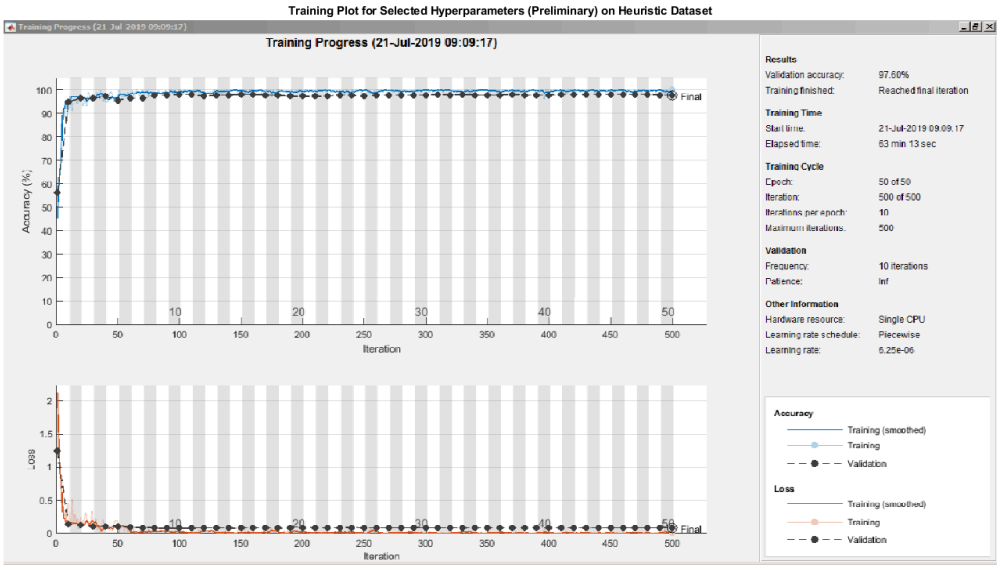


4.5: Train on heuristics dataset with selected preliminary hyperparameters

```
fileNameTemp = fullfile('./modelSelectedHeuInfo.mat');

% If the result struct file does not exists, conduct test; otherwise
 load
% the result struct file
if ~exist(fileNameTemp,'file')

    %seed random generator for consistency
    rng(0,'twister');

    % Configure selected preliminary training options
    selectedMiniBS = 100;
    selectedMaxEpochs = 50;
    selectedInitialLR = 1e-4;
    selectedLRSchedule = 'piecewise';
    selectedLRDropPeriod = 10;
    selectedLRDropFactor = 0.5;

    optionsHeuSelected = trainingOptions('sgdm', ...
        'MiniBatchSize',selectedMiniBS, ...
        'MaxEpochs',selectedMaxEpochs, ...
        'InitialLearnRate',selectedInitialLR, ...
        'LearnRateSchedule', selectedLRSchedule, ...
        'LearnRateDropPeriod', selectedLRDropPeriod, ...
        'LearnRateDropFactor', selectedLRDropFactor, ...
        'Shuffle','every-epoch', ...
        'ValidationData',validationSetHeu, ...
        'ValidationFrequency',10, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'OutputFcn',
 @(info)saveTrainingPlot(info, 'modelSelectedHeuTrainingPlot.png'));

    % close all figures if opened
    close(findall(groot, 'Type', 'Figure'));
    % train network
    [modelSelectedHeu, modelSelectedHeuInfo] =
 trainNetwork(trainingSetHeu,layersFull,optionsHeuSelected);

    % save network info
    fprintf("The validation accuracy with heuristics dataset is: %.2f
 %%\n", modelSelectedHeuInfo.ValidationAccuracy(end));
    save(fileNameTemp, 'modelSelectedHeuInfo');

else
    load(fileNameTemp,'modelSelectedHeuInfo');
    figure, imshow(imread(fullfile('./
modelSelectedHeuTrainingPlot.png')));
```

```
    title('Training Plot for Selected Hyperparameters (Preliminary) on
 Heuristic Dataset');
    fprintf("The validation accuracy with heuristics dataset is: %.2f
 %%\n", modelSelectedHeuInfo.ValidationAccuracy(end));
end
```

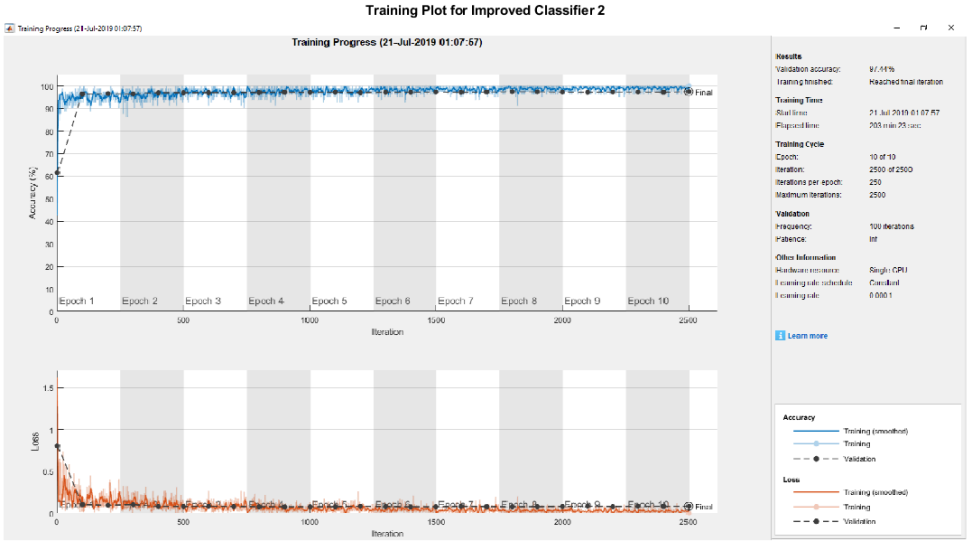
The validation accuracy with heuristics dataset is: 97.60 %



4.6: Train on full dataset with final selected hyperparameters

```
fileNameTemp = fullfile('./modelIC2Full.mat');

% If trained model file does not exists, train model; otherwise load
 model
if ~exist(fileNameTemp,'file')

    % Configure selected final training options
    selectedMiniBS = 80; %MiniBatchSize of 80 is used due to memory
 limitation of GPU on hand
    selectedMaxEpochs = 10;
    selectedInitialLR = 1e-4;

    optionsIC2Full = trainingOptions('sgdm', ...
        'MiniBatchSize',selectedMiniBS, ...
        'MaxEpochs',selectedMaxEpochs, ...
        'InitialLearnRate',selectedInitialLR, ...
        'Shuffle','every-epoch', ...
        'ValidationData',validationSetFull, ...
        'ValidationFrequency',100, ...
        'Verbose',false, ...
        'Plots','training-progress', ...
```

```
        'CheckPointPath',fullfile('./checkpoint'), ...
        'OutputFcn',
 @(info)saveTrainingPlot(info, 'modelIC2FullTrainingPlot.png'));

    % close all figures if opened
    close(findall(groot, 'Type', 'Figure'));
    % train network
    modelIC2Full =
 trainNetwork(trainingSetFull,layersFull,optionsIC2Full);

    % save network info
    save(fileNameTemp, 'modelIC2Full');

else
    load(fileNameTemp,'modelIC2Full');
    figure, imshow(imread(fullfile('./
modelIC2FullTrainingPlot.png')));
    title('Training Plot for Improved Classifier 2');
end
```



4.7: Classify the validation images using the fine-tuned network.

```
fileNameTemp = fullfile('./YPredIC2Full.mat');
if ~exist(fileNameTemp,'file')
    [YPredIC2Full,probsIC2Full] =
 classify(modelIC2Full,validationSetFull);
    save(fileNameTemp,'YPredIC2Full');
else
    load(fileNameTemp,'YPredIC2Full');
end
```

```
% Calculate the classification accuracy on the validation set.
% Accuracy is the fraction of labels that the network predicts
 correctly.
YValidationIC2Full = validationSetFull.Labels;
accuracyIC2Full = mean(YPredIC2Full == YValidationIC2Full);
fprintf("The validation accuracy for Improved Classifier 2 is: %.2f %%
\n", accuracyIC2Full * 100);
```

The validation accuracy for Improved Classifier 2 is: 97.44 %

Part 5: Performance Evaluation

This part evaluates each of the three classifiers: baseline classifiers, improved classifier 1 and improved classifier 2

5.1: Load test data

```
% Load full test dataset and build image store
dataFolderTestFull = './data/test';
imdsTestFull = imageDatastore(fullfile(dataFolderTestFull));
% Set the ImageDatastore ReadFcn
imdsTestFull.ReadFcn = @(filename)readAndPreprocessImage(filename);
```

5.2: Classify test data

```
fileNameTemp = fullfile('./ClassifierResult.mat');

% If classifier results do not exists, classify; otherwise load
 classifier
% results
if ~exist(fileNameTemp,'file')
    ClassifierResult = struct;
    % Baseline Classifier: classify the test images using the fine-
tuned network.
    ClassifierResult(1).name = 'Baseline Classifier';
    [ClassifierResult(1).YPred, ClassifierResult(1).probs] =
 classify(modelTransferFull,imdsTestFull);

    % Improved Classifier 1: classify the test images using the fine-
tuned network.
    ClassifierResult(2).name = 'Improved Classifier 1';
    [ClassifierResult(2).YPred, ClassifierResult(2).probs] =
 classify(modelAugFull,imdsTestFull);

    % Improved Classifier 2: classify the test images using the fine-
tuned network.
     ClassifierResult(3).name = 'Improved Classifier 2';
     [ClassifierResult(3).YPred, ClassifierResult(3).probs] =
 classify(modelIC2Full,imdsTestFull);

    % Ensemble of Classifiers
    ClassifierResult(4).name = 'Ensemble of Classifiers';
    for i = 1:2
```

```
        ClassifierResult(4).probs(:,i) =
 mean([ClassifierResult(1).probs(:,i), ...
            ClassifierResult(2).probs(:,i), ...
            ClassifierResult(3).probs(:,i)], 2);
    end
    ClassifierResult(4).YPred =
 categorical([ClassifierResult(4).probs(:,1) < 0.5],[0 1],
{'cat' 'dog'});

    save(fileNameTemp,'ClassifierResult');
else
    load(fileNameTemp,'ClassifierResult');
end
```

5.3: Save classified images into folder for manual checking and load manually classified labels

```
fileNameTemp = fullfile('./imageLabelTestSetGroundTruth.csv');

% Manually prepare the ground truth label file named
% "imageLabelTestSetGroundTruth.csv" with the first column as the
 image file name
% and the second column as the label. The order of the image file name
% should be preserved.

% If ground truth label does not exists, create prediction label for
 manual classification;
% otherwise load ground truth label
if ~exist(fileNameTemp,'file')

    % create cat folder if not exist
    if ~exist(fullfile('./data/test_cat'),'dir')
        mkdir(fullfile('./data/test_cat'))
    end
    % create dog folder if not exist
    if ~exist(fullfile('./data/test_dog'),'dir')
        mkdir(fullfile('./data/test_dog'))
    end
    for i = 1:size(imdsTestFull.Files,1)
        destfile = fullfile('./
data',strcat('test_',char(ClassifierResult(4).YPred(i))));
        copyfile (char(imdsTestFull.Files(i)), destfile);
    end

    % export image labels for manual checking
    imageLabel = cell(size(imdsTestFull.Files,1),2);
    for i = 1: size(imdsTestFull.Files,1)
        [~,imageName,imageExt] =
 fileparts(char(imdsTestFull.Files(i)));
        imageLabel(i,1) = {strcat(imageName,imageExt)};
```

```
        imageLabel(i,2) = {char(ClassifierResult(4).YPred(i))};
    end
    writecell(imageLabel,fullfile('./imageLabelTestSetPred.csv'));
else
    imageLabel = cell(size(imdsTestFull.Files,1),1);
    for i = 1: size(imdsTestFull.Files,1)
        [~,imageName,imageExt] =
 fileparts(char(imdsTestFull.Files(i)));
        imageLabel(i,1) = {strcat(imageName,imageExt)};
    end

    imageLabelTestSetGroundTruth = readcell(fileNameTemp);
    % check if the image file names are in the correct order
    if isequal(imageLabel(:,1),imageLabelTestSetGroundTruth(:,1))
        % read label into ImageDateStore
        imdsTestFull.Labels =
 categorical(imageLabelTestSetGroundTruth(:,2));
    % if not, check if the label file names match the imageDatestore
 file names
    elseif
 size(intersect(imageLabel(:,1),imageLabelTestSetGroundTruth(:,1)),1)
 == size(imageLabel(:,1),1)
        tempLabels = cell(size(imageLabel(:,1)));
        for i = 1:size(imageLabel(:,1),1)
            for j = 1:size(imageLabelTestSetGroundTruth(:,1),1)
                if
 isequal(imageLabel(i,1),imageLabelTestSetGroundTruth(j,1))
                    tempLabels(i,1)=
 imageLabelTestSetGroundTruth(j,2);
                    imageLabelTestSetGroundTruth(j,:)=[];
                    break;
                end
            end
        end
        imdsTestFull.Labels = categorical(tempLabels);
    else % if label does not match imageDatestore file names
        fprintf("Ground truth labels do not match imageDatastore
 files!!!");
    end
end
```

5.4: Calculate the classification accuracy on the test set and display sample images

```
YGroundTruthLabelTestFull = imdsTestFull.Labels;

% extract indices of ground truth cats and dogs
idxGroundTruthCat = find(YGroundTruthLabelTestFull == 'cat');
idxGroundTruthDog = find(YGroundTruthLabelTestFull == 'dog');

%reset imageDataStore read function
imdsTestFull.ReadFcn = @(filename)imread(filename);
```

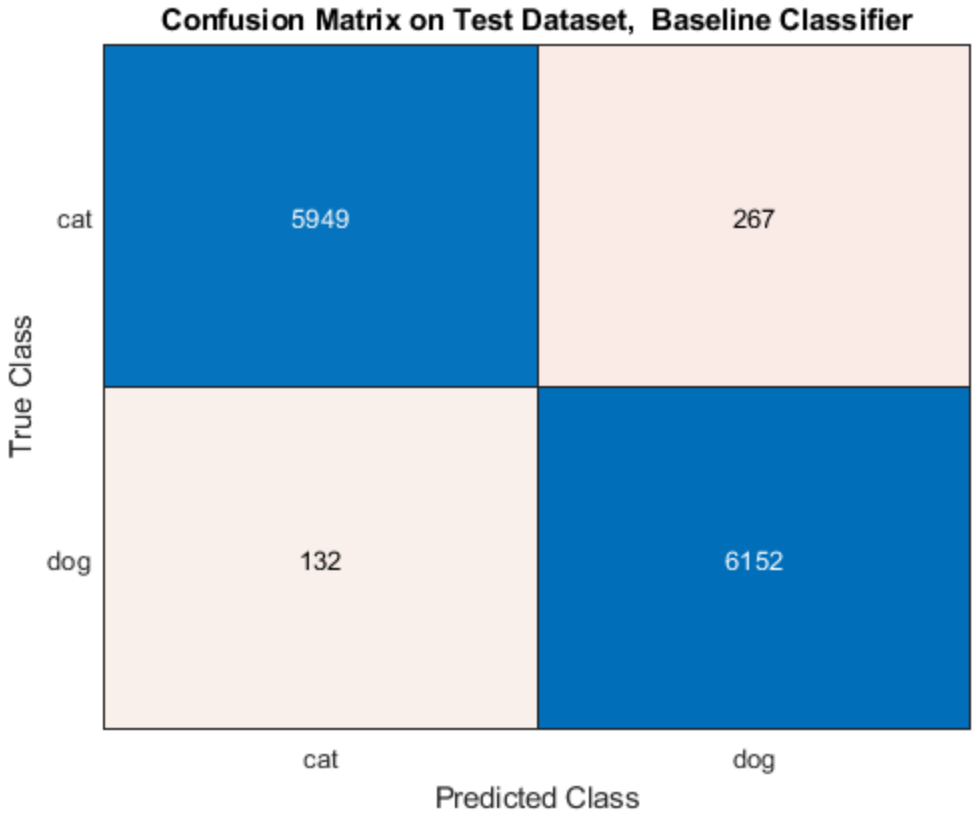
```
for i = 1:size(ClassifierResult,2)
    ClassifierResult(i).accuracy = mean(ClassifierResult(i).YPred ==
 YGroundTruthLabelTestFull);
    fprintf("The classification accuracy for %s is: %.2f %%\n",
 ClassifierResult(i).name, ...
        ClassifierResult(i).accuracy * 100);
    ClassifierResult(i).confMat =
 confusionmat(YGroundTruthLabelTestFull,ClassifierResult(i).YPred);
    % Display confusion chart
    figure, confusionchart(ClassifierResult(i).confMat,{'cat','dog'});
    title(strcat("Confusion Matrix on Test Dataset,  ",
 ClassifierResult(i).name));

    % Compute the ROC curve, positive class assumes to be dog
    [X,Y,T,AUC] = perfcurve(YGroundTruthLabelTestFull,
 ClassifierResult(i).probs(:,2), 'dog');

    % Plot the ROC curve
    figure, plot(X,Y)
    xlabel('False positive (dog) rate'); ylabel('True positive (dog)
 rate');
    title(strcat("ROC Curves on Test Dataset,  ",
 ClassifierResult(i).name));

    % Display the area under the curve.
    fprintf("Area under curve for %s is: %.4f \n",
 ClassifierResult(i).name, AUC)

    % Display montage of correctly identified images
    % sorted indices by probability
    [~,idxSortByCat] = sort(ClassifierResult(i).probs(:,1));
    [~,idxSortByDog] = sort(ClassifierResult(i).probs(:,2));

    % indices of correct and incorrect classifications
    idxCorrectPred = find (ClassifierResult(i).YPred ==
 YGroundTruthLabelTestFull);
    idxIncorrectPred = find (~(ClassifierResult(i).YPred ==
 YGroundTruthLabelTestFull));

    % indices of correctly classified cats and dogs
    idxCorrectCatPred = intersect(idxGroundTruthCat,idxCorrectPred);
    idxCorrectDogPred = intersect(idxGroundTruthDog,idxCorrectPred);

    % indices of cats incorrectly classified as dogs
    idxIncorrectCatPred =
 intersect(idxGroundTruthCat,idxIncorrectPred);

    % indices of dogs incorrectly classified as cats
    idxIncorrectDogPred =
 intersect(idxGroundTruthDog,idxIncorrectPred);

    % display sample images correctly classified as cats with the
 highest confidence
```

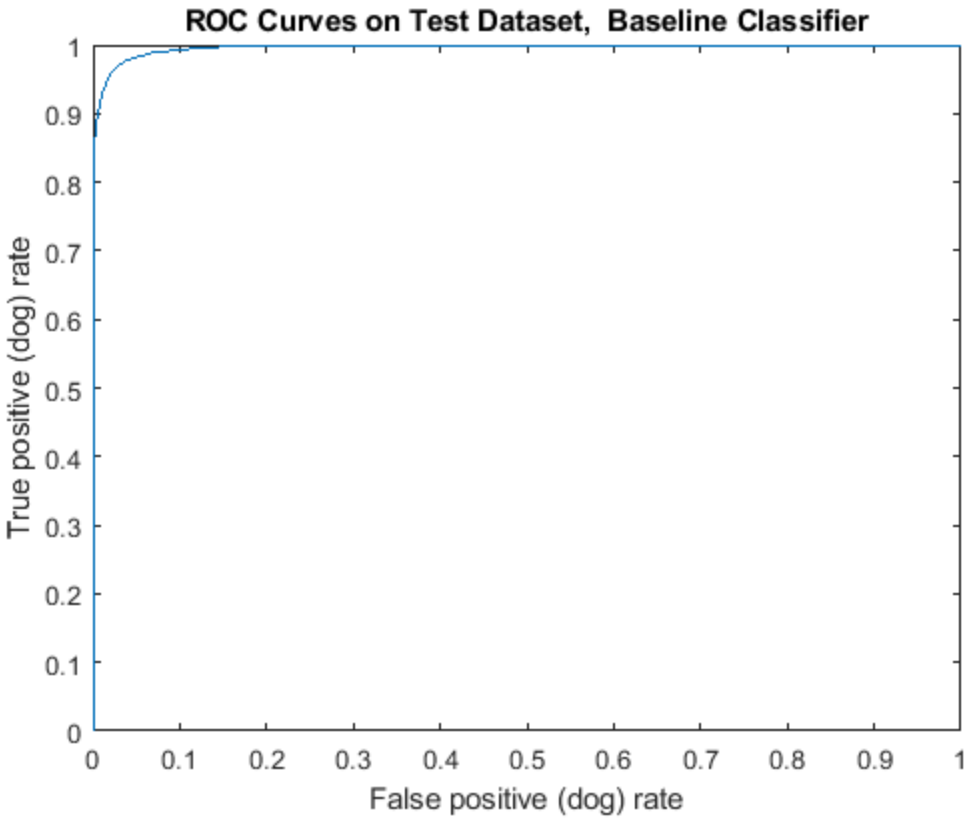
```
   subIdxCorrectCat =
intersect(idxSortByCat,idxCorrectCatPred, 'stable');
   if ~isempty(subIdxCorrectCat)
       subIdxCorrectCat = subIdxCorrectCat
(max(size(subIdxCorrectCat,1) - 5,1): end);
       figure, montage(subset(imdsTestFull, subIdxCorrectCat));
       % construct confidence % of the images
       lowerRange = ClassifierResult(i).probs(subIdxCorrectCat(1),
1);
       upperRange = ClassifierResult(i).probs(subIdxCorrectCat(end),
1);
       if lowerRange == upperRange
           confidenceStr = strcat(num2str(lowerRange *
100, '%.2f'), "%");
       else
           confidenceStr = strcat(num2str(lowerRange *
100, '%.2f'), "% to ", ...
               num2str(upperRange * 100, '%.2f'), "%");
       end
       title(strcat("Sample - Correctly Classified as Cats, ",
ClassifierResult(i).name, ...
           " | Confidence: ", confidenceStr));
   end

   % display sample images correctly classified as dogs with the
highest confidence
   subIdxCorrectDog =
intersect(idxSortByDog,idxCorrectDogPred, 'stable');
   if ~isempty(subIdxCorrectDog)
       subIdxCorrectDog = subIdxCorrectDog
(max(size(subIdxCorrectDog,1) - 5,1): end);
       figure, montage(subset(imdsTestFull, subIdxCorrectDog));
       % construct confidence % of the images
       lowerRange = ClassifierResult(i).probs(subIdxCorrectDog(1),
2);
       upperRange = ClassifierResult(i).probs(subIdxCorrectDog(end),
2);
       if lowerRange == upperRange
           confidenceStr = strcat(num2str(lowerRange *
100, '%.2f'), "%");
       else
           confidenceStr = strcat(num2str(lowerRange *
100, '%.2f'), "% to ", ...
               num2str(upperRange * 100, '%.2f'), "%");
       end
       title(strcat("Sample - Correctly Classified as Dogs, ",
ClassifierResult(i).name, ...
           " | Confidence: ", confidenceStr));
   end

   % display sample images incorrectly classified as dogs with the
lowest confidence
   subIdxIncorrectCat =
intersect(idxSortByDog,idxIncorrectCatPred, 'stable');
```

```
    if ~isempty(subIdxIncorrectCat)
        subIdxIncorrectCat = subIdxIncorrectCat (1:
 min(size(subIdxIncorrectCat,1), 6));
        figure, montage(subset(imdsTestFull, subIdxIncorrectCat));
        % construct confidence % of the images
        lowerRange = ClassifierResult(i).probs(subIdxIncorrectCat(1),
 2);
        upperRange =
 ClassifierResult(i).probs(subIdxIncorrectCat(end), 2);
        if lowerRange == upperRange
            confidenceStr = strcat(num2str(lowerRange *
 100, '%.2f'), "%");
        else
            confidenceStr = strcat(num2str(lowerRange *
 100, '%.2f'), "% to ", ...
                num2str(upperRange * 100, '%.2f'), "%");
        end
        title(strcat("Sample - Incorrectly Classified as Dogs (Should
 be Cats), ", ClassifierResult(i).name, ...
            " | Confidence: ", confidenceStr));
    end

    % display sample images incorrectly classified as cats with the
 lowest confidence
    subIdxIncorrectDog =
 intersect(idxSortByCat,idxIncorrectDogPred, 'stable');
    if ~isempty(subIdxIncorrectDog)
        subIdxIncorrectDog = subIdxIncorrectDog (1:
 min(size(subIdxIncorrectDog,1), 6));
        figure, montage(subset(imdsTestFull, subIdxIncorrectDog));
        % construct confidence % of the images
        lowerRange = ClassifierResult(i).probs(subIdxIncorrectDog(1),
 1);
        upperRange =
 ClassifierResult(i).probs(subIdxIncorrectDog(end), 1);
        if lowerRange == upperRange
            confidenceStr = strcat(num2str(lowerRange *
 100, '%.2f'), "%");
        else
            confidenceStr = strcat(num2str(lowerRange *
 100, '%.2f'), "% to ", ...
                num2str(upperRange * 100, '%.2f'), "%");
        end
        title(strcat("Sample - Incorrectly Classified as Cats (Should
 be Dogs), ", ClassifierResult(i).name, ...
            " | Confidence: ", confidenceStr));
    end
end
```

```
The classification accuracy for Baseline Classifier is: 96.81 %
Area under curve for Baseline Classifier is: 0.9964
The classification accuracy for Improved Classifier 1 is: 96.93 %
Area under curve for Improved Classifier 1 is: 0.9963
The classification accuracy for Improved Classifier 2 is: 97.70 %
```

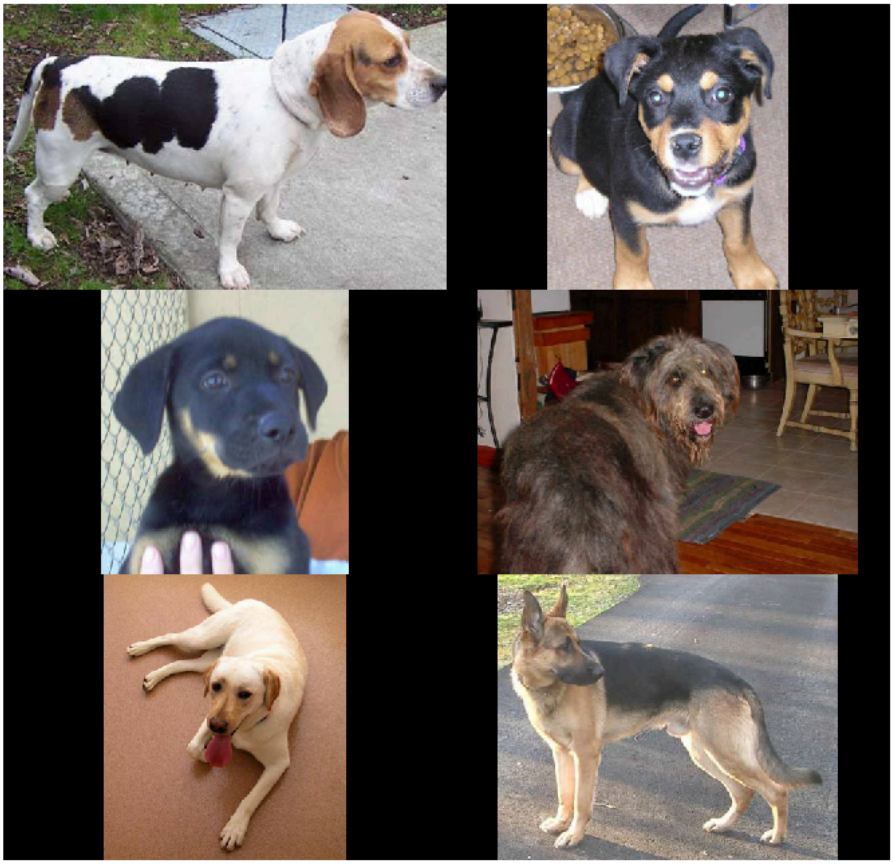
```
Area under curve for Improved Classifier 2 is: 0.9978
The classification accuracy for Ensemble of Classifiers is: 97.88 %
Area under curve for Ensemble of Classifiers is: 0.9978
```




ROC Curves on Test Dataset, Baseline Classifier
True positive (dog) rate
False positive (dog) rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

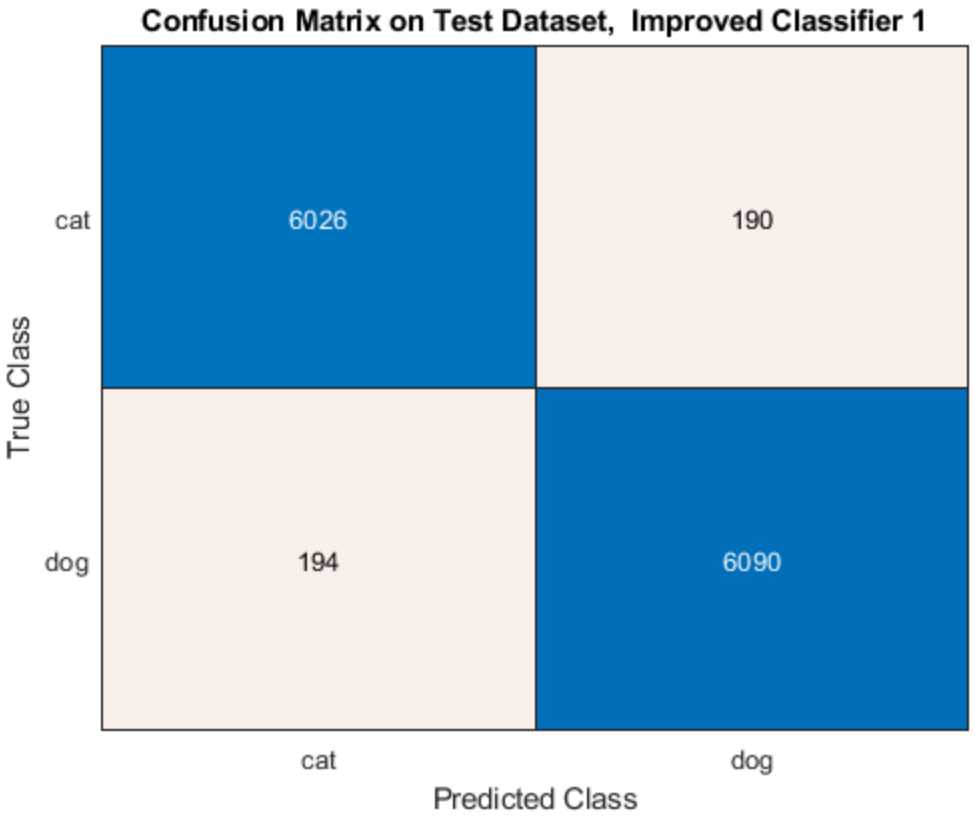

Sample - Correctly Classified as Cats, Baseline Classifier | Confidence: 100.00%

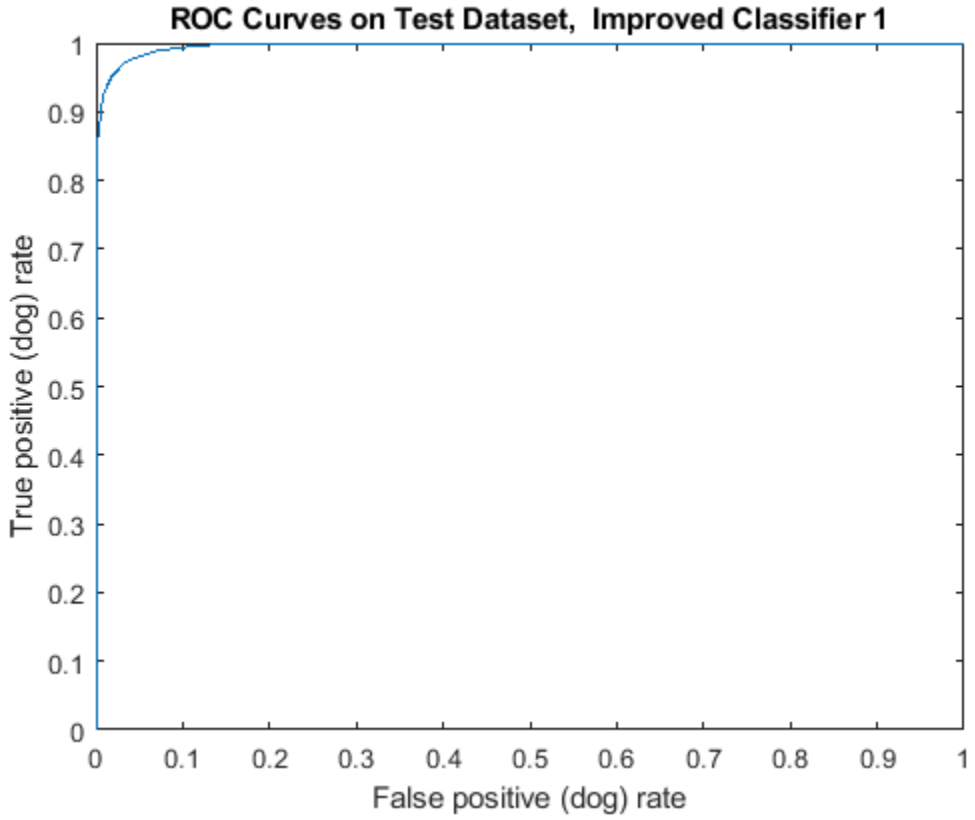
Sample - Correctly Classified as Dogs, Baseline Classifier | Confidence: 100.00%



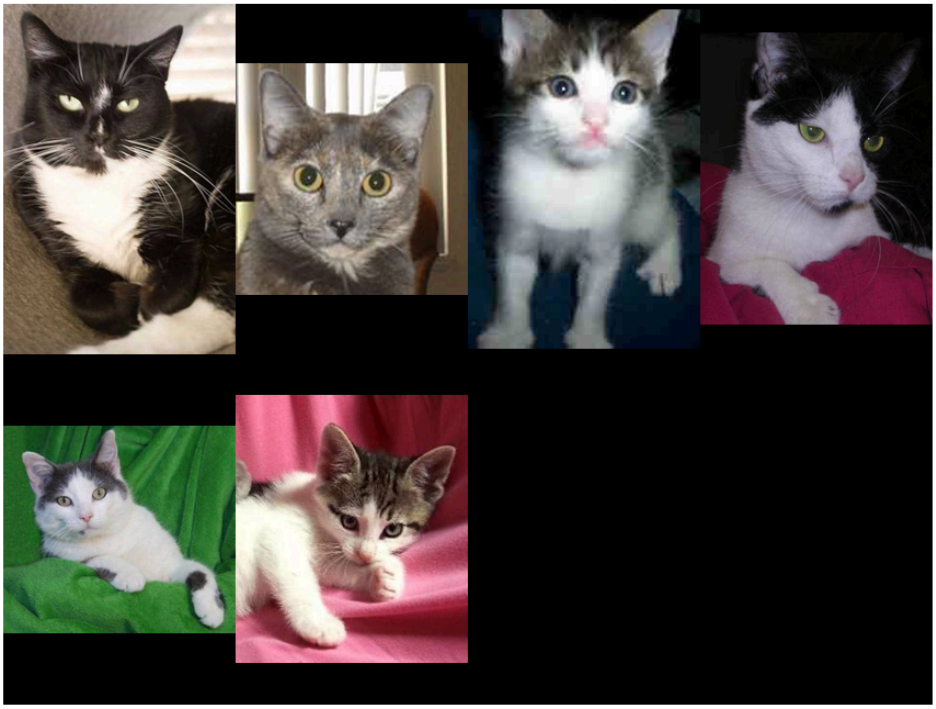
Sample - Incorrectly Classified as Dogs (Should be Cats), Baseline Classifier | Confidence: 50.07% to 50.76%

Sample - Incorrectly Classified as Cats (Should be Dogs), Baseline Classifier | Confidence: 51.03% to 52.91%


Confusion Matrix on Test Dataset, Improved Classifier 1
cat
dog
6026
190
194
6090
True Class
cat
dog
Predicted Class

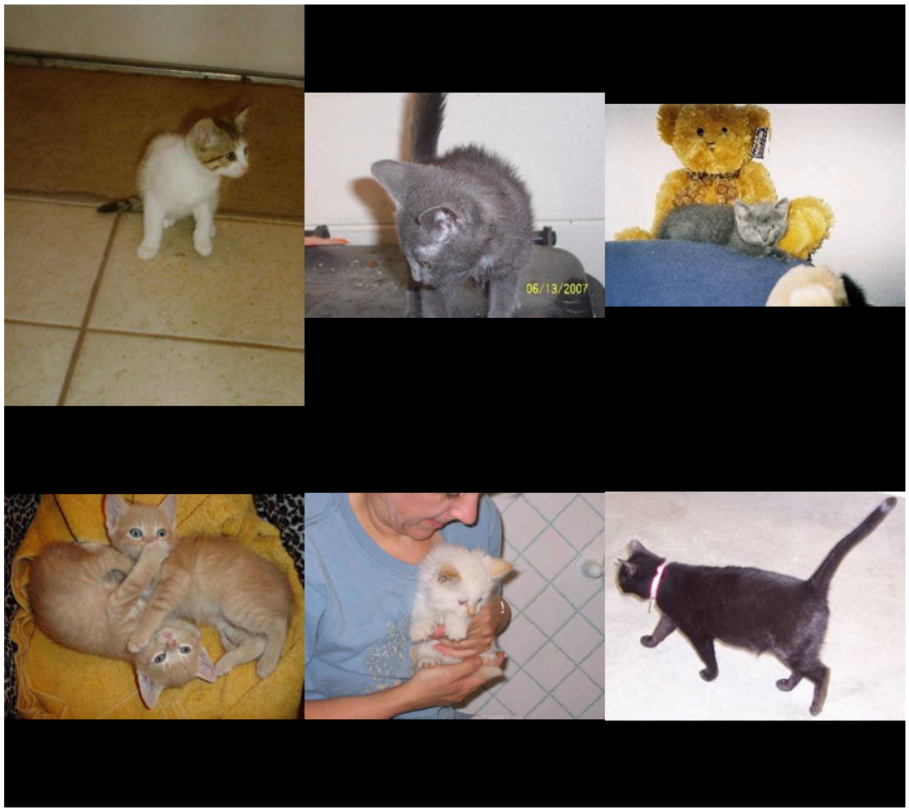

ROC Curves on Test Dataset, Improved Classifier 1
True positive (dog) rate
False positive (dog) rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Sample - Correctly Classified as Cats, Improved Classifier 1 | Confidence: 100.00%

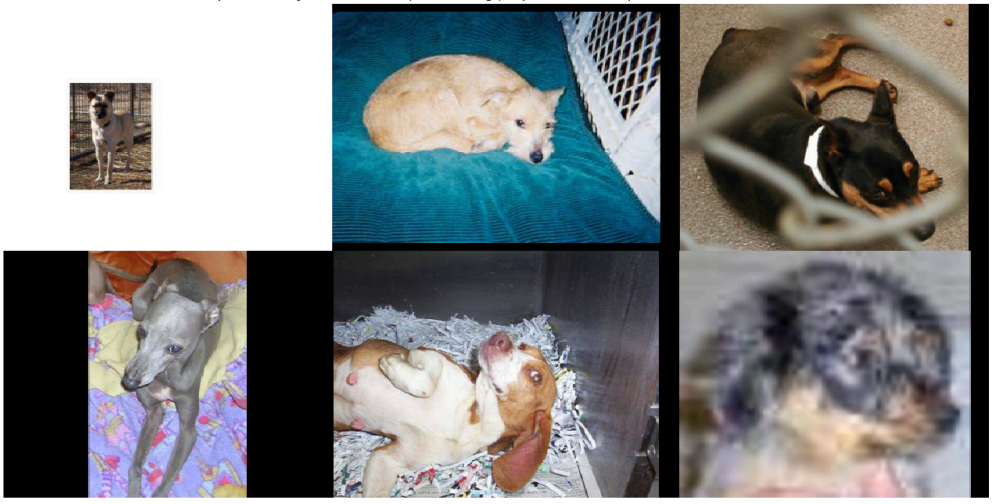


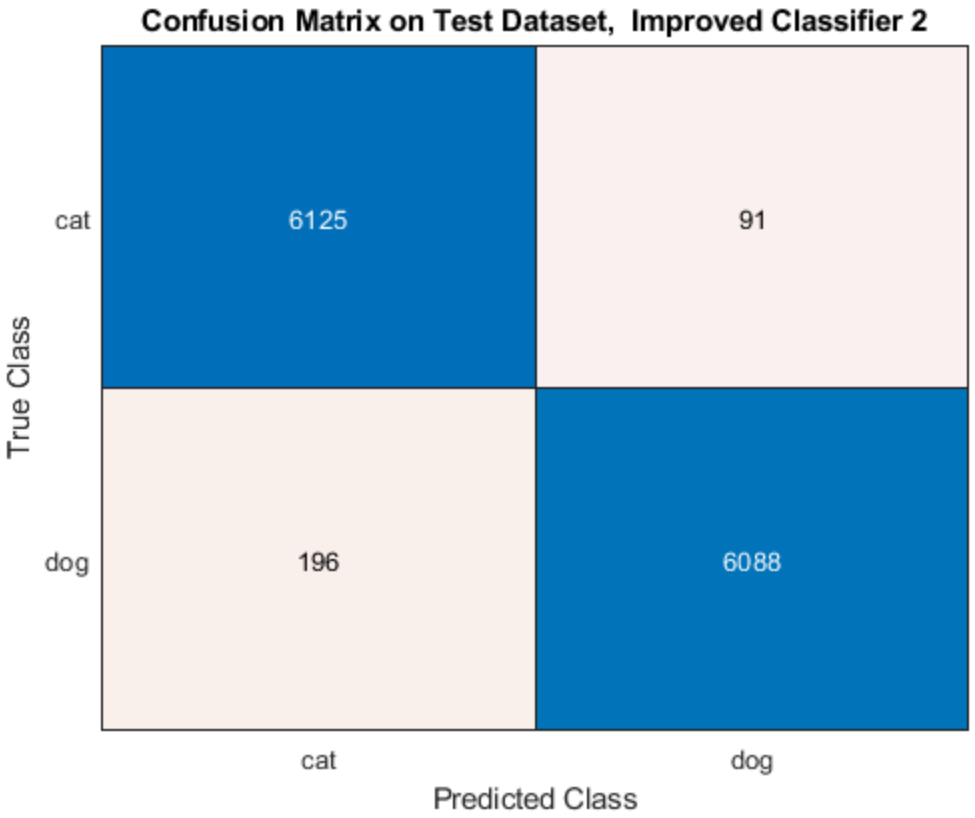
Sample - Correctly Classified as Dogs, Improved Classifier 1 | Confidence: 100.00%

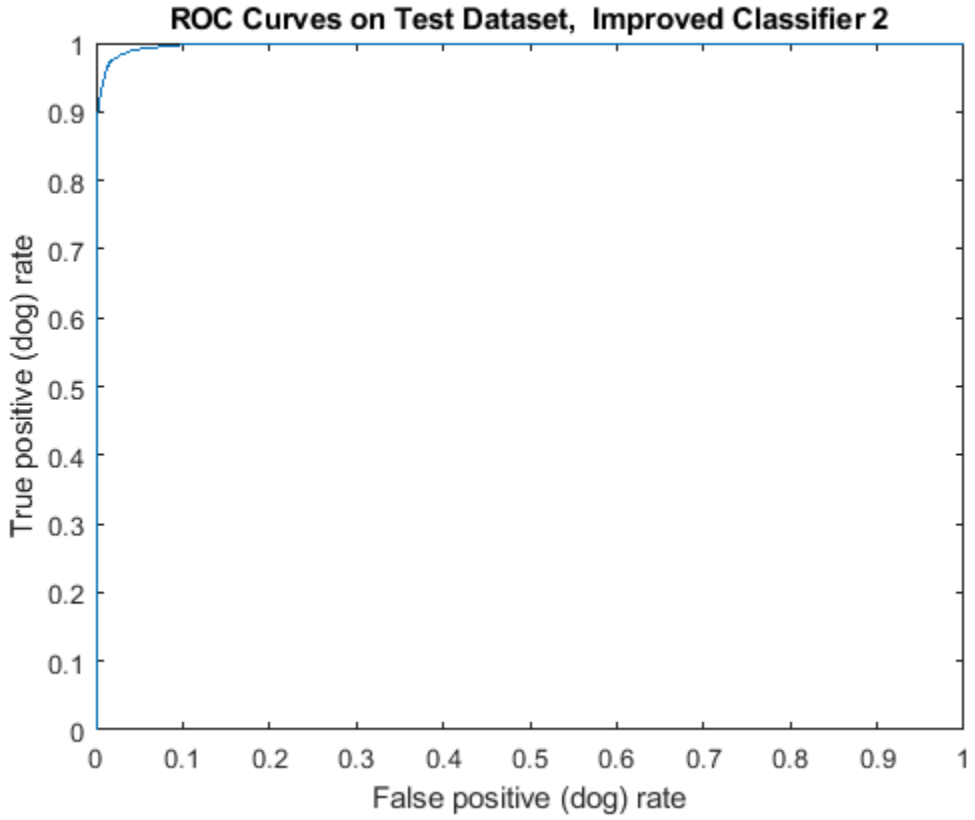
Sample - Incorrectly Classified as Dogs (Should be Cats), Improved Classifier 1 | Confidence: 50.20% to 52.41%



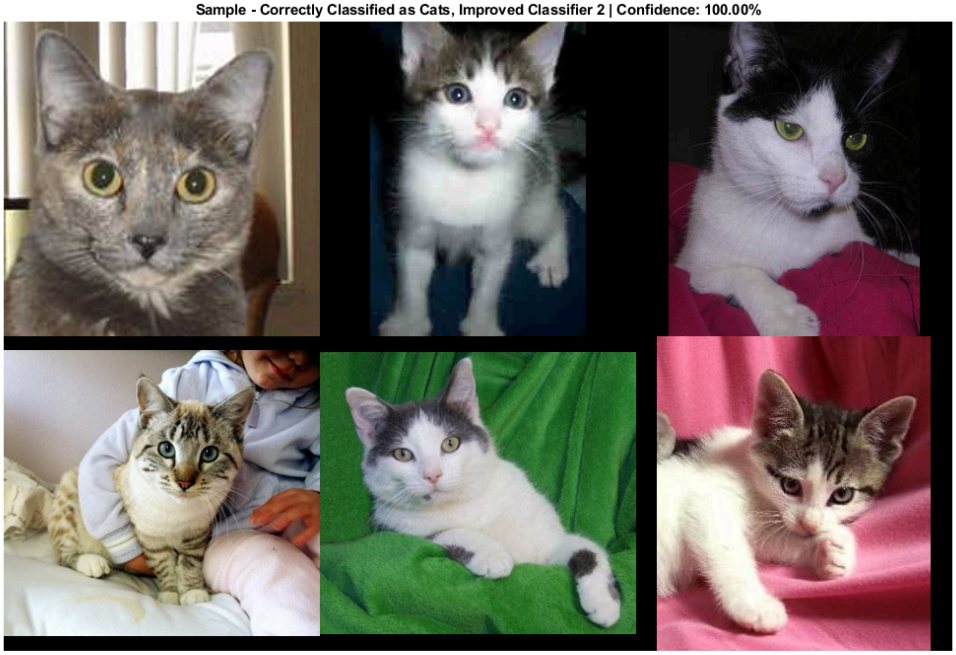
Sample - Incorrectly Classified as Cats (Should be Dogs), Improved Classifier 1 | Confidence: 50.31% to 50.92%




Confusion Matrix on Test Dataset, Improved Classifier 2
cat
dog
6125
91
196
6088
True Class
cat
dog
Predicted Class


ROC Curves on Test Dataset, Improved Classifier 2
True positive (dog) rate
False positive (dog) rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

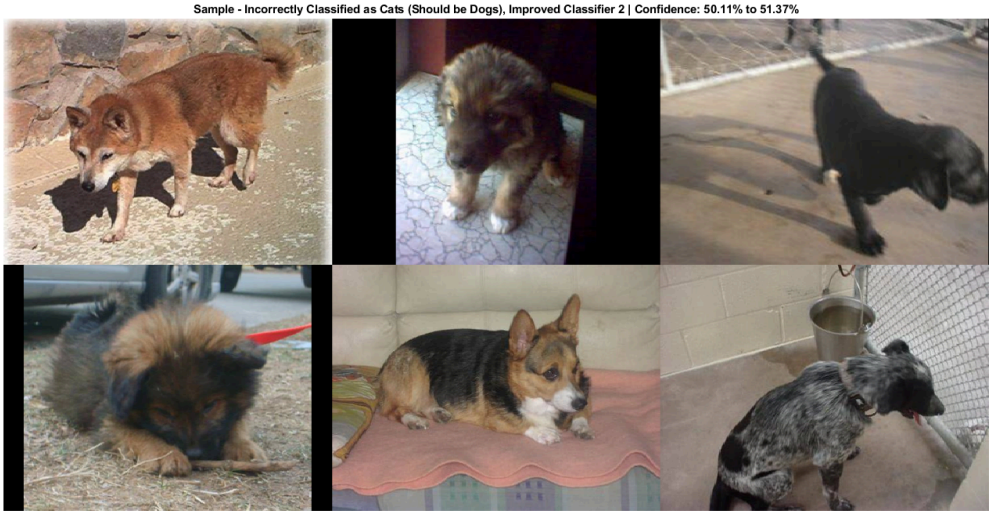
Sample - Correctly Classified as Cats, Improved Classifier 2 | Confidence: 100.00%

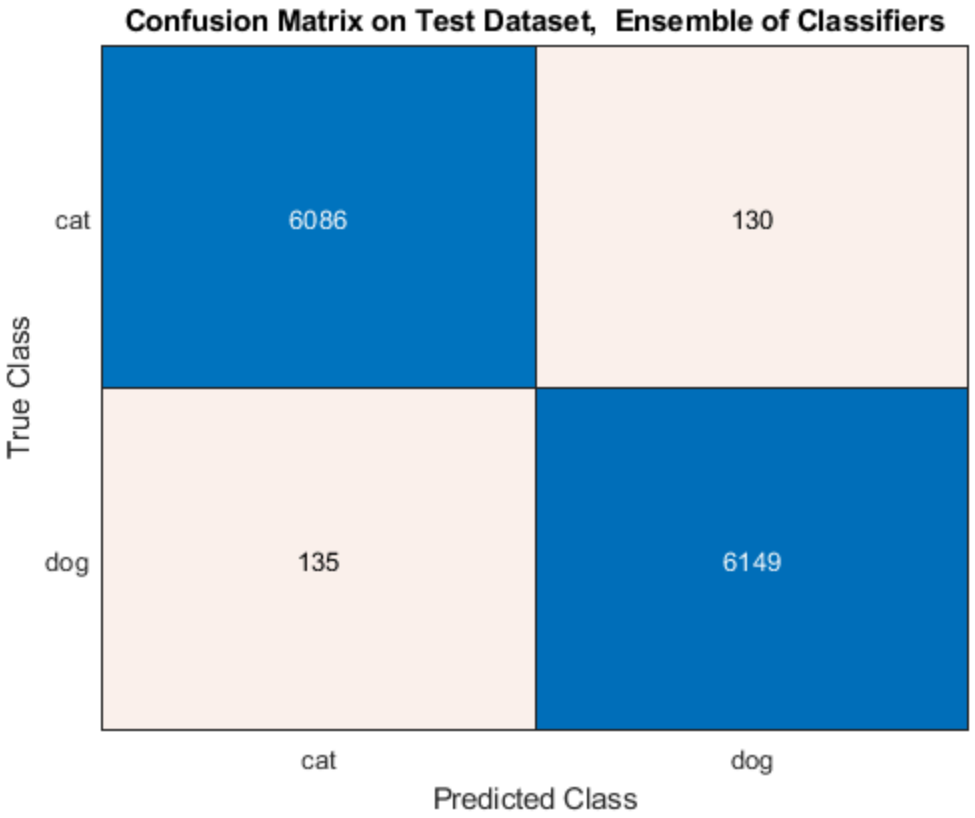


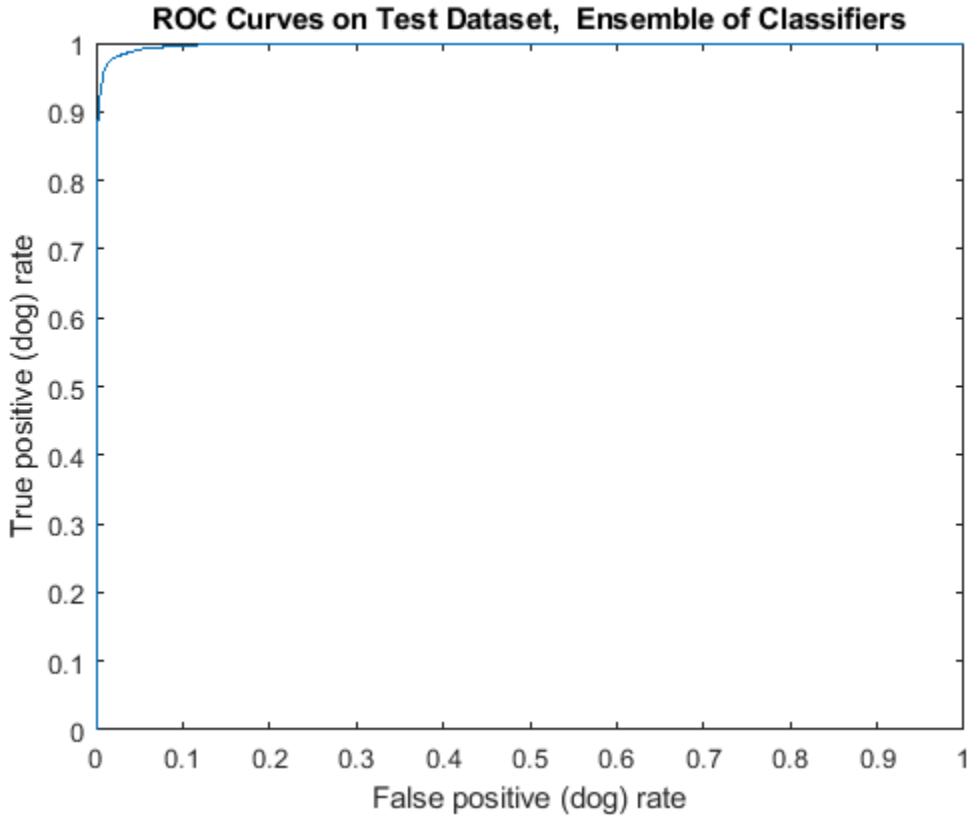
Sample - Correctly Classified as Dogs, Improved Classifier 2 | Confidence: 100.00%

Sample - Incorrectly Classified as Dogs (Should be Cats), Improved Classifier 2 | Confidence: 50.57% to 57.10%

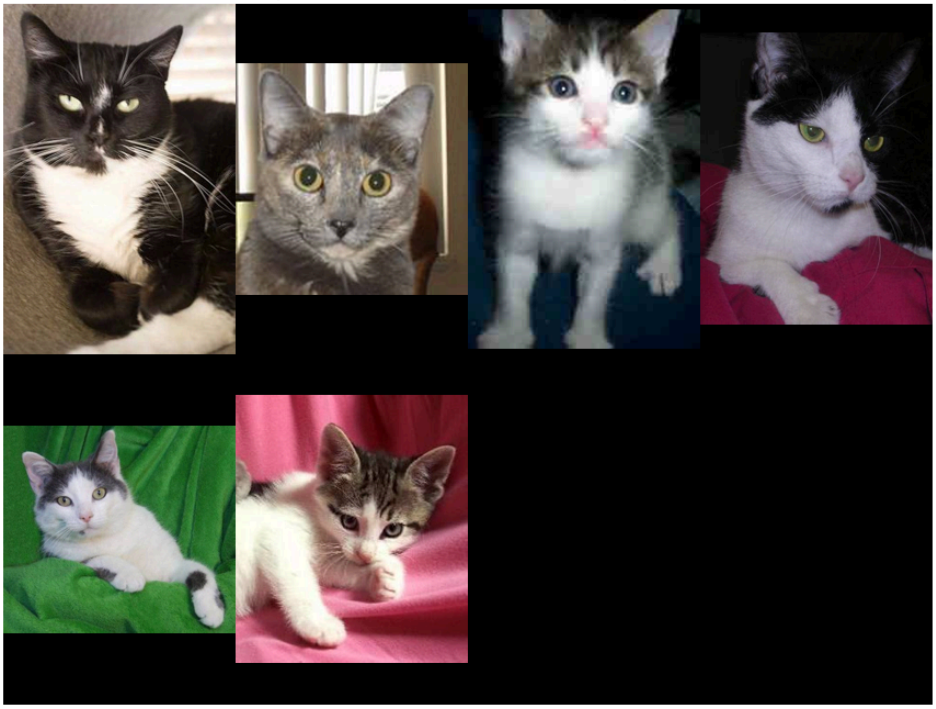
Sample - Incorrectly Classified as Cats (Should be Dogs), Improved Classifier 2 | Confidence: 50.11% to 51.37%



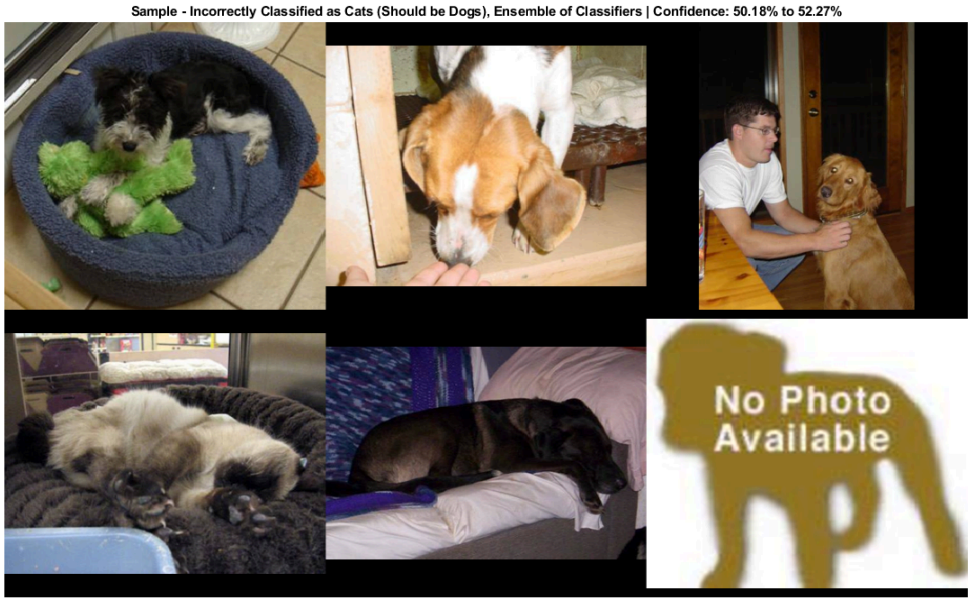

Confusion Matrix on Test Dataset, Ensemble of Classifiers
cat
dog
6086
130
135
6149
True Class
cat
dog
Predicted Class


ROC Curves on Test Dataset, Ensemble of Classifiers
True positive (dog) rate
False positive (dog) rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Sample - Correctly Classified as Cats, Ensemble of Classifiers | Confidence: 100.00%



Sample - Correctly Classified as Dogs, Ensemble of Classifiers | Confidence: 100.00%



Auxiliary Function

This function saves the training plot of a CNN at the end of the training. Please close all figures before training to avoid errors.

```
function stop = saveTrainingPlot(info, filename)
stop = false;  %prevents this function from ending trainNetwork
 prematurely
if info.State == 'done'   %check if all iterations have completed
% if true
    tempFigHandle = findall(groot, 'Type', 'Figure');
    set(tempFigHandle(1), 'Position', get(0, 'Screensize'));
    saveas(tempFigHandle(1),fullfile('./', filename));
end
end
```

Published with MATLAB® R2019a