Final Project Report

Comparison of Pretrained Networks for Image Classification

SongJalern Kamonpurn (王佩佩) P86107061

Luigi Gan (蔡嘉煌) P86107053

Course

111 Computer Aided Engineering

Biomedical Engineering Department

Table of Contents

Problem of statement

Image classification is a vital task in the field of computer vision, with numerous applications in diverse domains. Accurate image classification can enable autonomous vehicles to recognize traffic signs and pedestrians, assist in the diagnosis of medical images, and aid in the organization and retrieval of large image databases. [1] The goal of this project is to develop an app for classifying images into three categories: dogs, cats, and neither dogs nor cats. The app will use pretrained networks as the basis for the classification model and compare the performance of different models. The accuracy and confusion matrix of the classification results will be measured, and the best performing model will be identified. The app will also be able to handle real-time classification of images. This is a challenging task because the images may vary in terms of lighting, pose, and background, and the animals may appear in different sizes and at different distances from the camera.

Materials

To achieve the objectives of this project, we used Dataset below:

1. Cat and Dog images retrieved from Kaggle Dog vs Cat dataset [2]

   They've provided Microsoft Research with over three million images of cats and dogs, manually classified by people at thousands of animal shelters across the United States. Kaggle is fortunate to offer a subset of this data for fun and research.

2. NotCatorDog images retrieved from Kaggle Caltech 256 Image Dataset [3]

   There are 30,607 images in this dataset spanning 257 object categories. Object categories are extremely diverse, ranging from grasshopper to tuning fork.

3. Convolutional neural network

   - ResNet50 [4]

     is a CNN that was developed by researchers at Microsoft. It was trained on the ImageNet dataset, which contains over 14 million images and 1000 different object categories.

   - GoogLeNet [5]

     is a CNN that was developed by researchers at Google. It was also trained on the ImageNet dataset and is known for its efficiency in terms of the number of parameters it has compared to other networks.

   - AlexNet [6]

     is a CNN that was developed by researchers at the University of Toronto. It was the first CNN to achieve significant results in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and is known for its use of rectified linear units (ReLUs) as activation functions.

Methods

To achieve the objectives of this project, we followed the steps outlined below [6]:

1. Data preparation: We used a dataset of 2400 images, 800 images for each class. The images were resized to 224x224 pixels, 227x227 pixels and normalized to the range [0, 1]. The data was split into a training set (1920 images) and a validation set (480 images).
2. Model selection: We selected the ResNet50, GoogleNet, and AlexNet networks as pretrained models for our classification task.
3. Network modification: We modified the final layers of the networks to fit the number of classes in our dataset (3). We replaced the fully connected and classification layers with new layers and froze the weights of the first 10 layers of the networks.
4. Data augmentation: To improve the generalization of the models, we applied random image transformations such as reflection and scaling to the training data using the **imageDataAugmenter** object.
5. Training: We trained the modified ResNet50, GoogleNet, and AlexNet networks and the categorical cross-entropy loss function. The network was trained for 10 epochs with a mini-batch size of 10 and a validation frequency of 20.
6. APP: we used results networks of ResNet50, GoogleNet, and AlexNet networks to develop an app in MATLAB APP designer for classifying images into three categories: dogs, cats, and neither dogs nor cats with accuracy

Results and discussion

We trained the modified networks on the training set and evaluated their performance on the validation set during training. The results showed that all three networks achieved high accuracy on the validation set, with ResNet50 and GoogLeNet achieving the high accuracy as show in Table 1

Table 1: Classification Accuracy for each network.

Network	**Accuracy**	**Time**
ResNet50	0.9896	69 min 30 sec
GoogLeNet	0.9854	26 min 30 sec
AlexNet	0.9521	12 min 35 sec

After training, we used the trained networks to classify images and compared the results. The results showed that all three networks achieved high accuracy, with GooLeNet performing the best overall.

The confusion matrices for the three networks are shown in Figure 1. These matrices show the number of images that were correctly and incorrectly classified by each network. From the matrices, we can see that all three networks had high accuracy, with a relatively low number of misclassifications.

it seems that the ResNet50 and GoogLeNet networks performed better than AlexNet in terms of classification accuracy. The ResNet50 network had an accuracy of 98.96%, while GoogLeNet had an accuracy of 98.54%. AlexNet had the lowest accuracy among the three networks, with 95.21%.

In terms of training time, AlexNet required the least amount of time, followed by GoogLeNet, and finally ResNet50. These results indicate that GoogLeNet may be the most efficient network for this task, while still maintaining a high level of accuracy.

ResNet50 GoogLeNet AlexNet

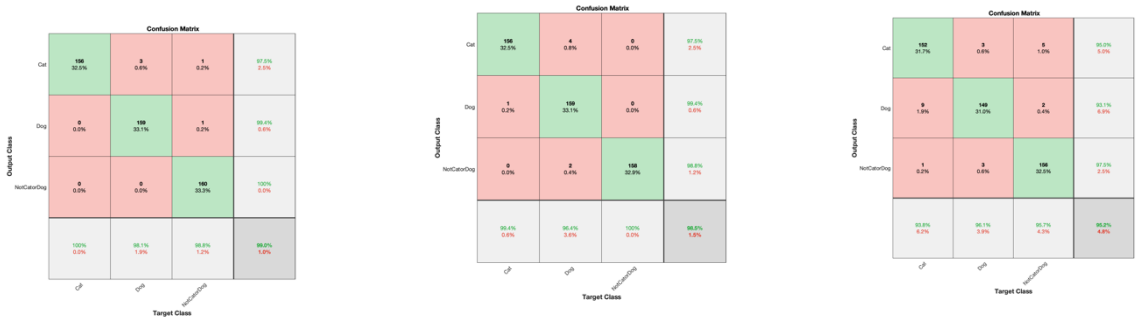
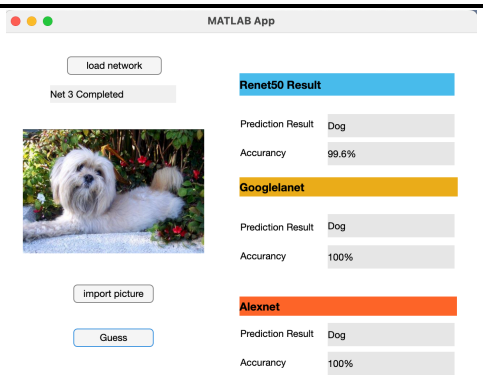
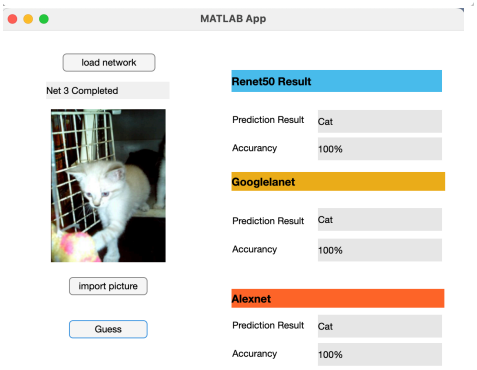
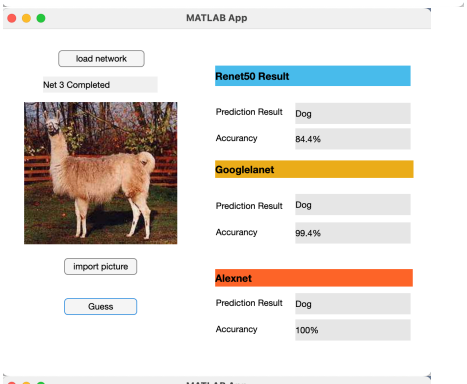
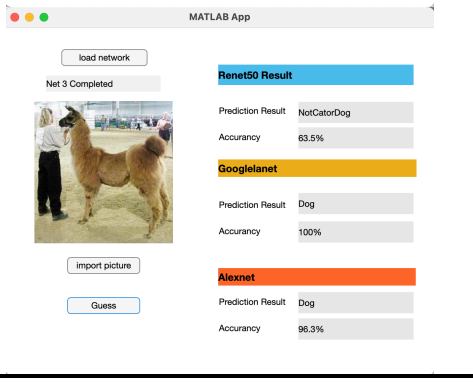
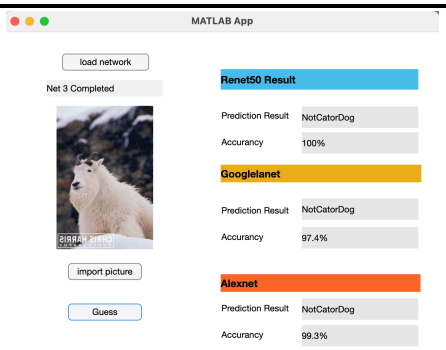


Figure 1: Confusion Matrices for the Three Networks

The results of using the three convolutional neural networks (CNNs) - ResNet50, GoogLeNet, and AlexNet - for image classification on the APP show that all three networks achieved high accuracy when the input image was of a dog or cat. However, when the input image had features similar to a dog or cat, such as four legs, a tail, or fur, the results were less accurate. This can be seen in Table 2, where the accuracy of the networks in classifying lamas is lower than the accuracy in classifying dogs and cats. It is important to note that the accuracy of these networks may be affected by the specific characteristics of the input image, and further optimization may be necessary to improve their performance.

Table 2 : Example results from APP

Type	APP results
Dog	
Cat	
Lama	 

Goat	

Conclusion

In this case, the ResNet50 and GoogLeNet networks were able to effectively learn and classify images of cats and dogs, These results suggest that both the ResNet50 and GoogLeNet networks are effective at classifying images of cats and dogs, with the ResNet50 network performing slightly better. On the other hand, the AlexNet network performed significantly worse, indicating that it may not be as well-suited for this task. these results may vary depending on the specific training and test data used, as well as the training and evaluation procedures.

Bibliography

[1] [Online]. Available: https://kili-technology.com/data-labeling/computer-vision/image-annotation/image-recognition-with-machine-learning-how-and-why.

[2] [Online]. Available: https://www.kaggle.com/c/dogs-vs-cats .

[3] [Online]. Available: https://www.kaggle.com/datasets/jessicali9530/caltech256.

[4] [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/resnet50.html.

[5] [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/googlenet.html.

[6] [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/alexnet.html.

[7] [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html.

Code

Load and store image

```
% Location
path = '/Users/kmp/Desktop/Com-aid final project';
% Store the output
imageFolder = fullfile(path,'Dataset');
imds = imageDatastore(imageFolder, 'LabelSource', 'foldernames',
'IncludeSubfolders',true);
```

recheck images and label of each category

```
% Find the first instance of an image for each category
Dog = find(imds.Labels == 'Dog', 1);
Cat = find(imds.Labels == 'Cat', 1);
NotCatorDog=find(imds.Labels=='NotCatorDog',1);
figure
imshow(readimage(imds,Dog))
```

```
imshow(readimage(imds,Cat))
```

```
imshow(readimage(imds,NotCatorDog))
```

Set number of each label to be equal(800)

```
% count each label
tbl = countEachLabel(imds)
```

```
tbl = 3×2 table
```

	Label	Count
1	Cat	989
2	Dog	990
3	NotCatorDog	1088

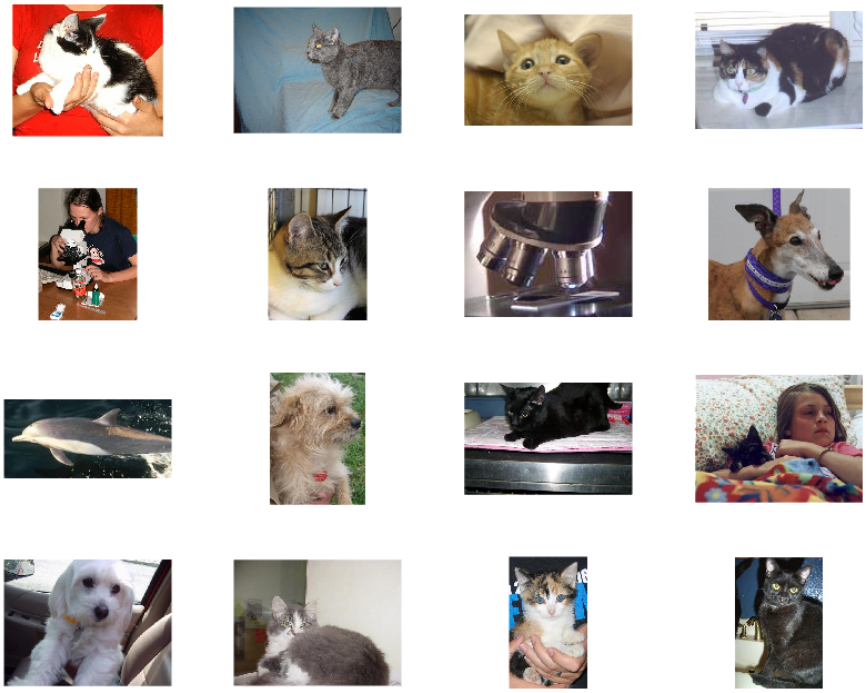
```
% Determine the smallest amount of images in a category
minSetCount = min(tbl{:,2});
% Limit the number of images to reduce the time it takes
% run this example.
maxNumImages = 800;
minSetCount = min(maxNumImages,minSetCount);

% 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)
```

```
ans = 3×2 table
```

	Label	Count
1	Cat	800
2	Dog	800
3	NotCatorDog	800

Divide data into training and validation data set

```
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.8,'randomized');
% Display some sample images.
numTrainImages = numel(imdsTrain.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(imdsTrain,idx(i));
    imshow(I)
end
```



```
% set number of classes
numClasses = numel(categories(imdsTrain.Labels))
```

```
numClasses = 3
```

Load Pretrained Network

resnet50

```
net1=resnet50;
net1.Layers(1)
```

```
ans =
```

```
        ImageInputLayer with properties:

                            Name: 'input_1'
                       InputSize: [224 224 3]
                SplitComplexInputs: 0

         Hyperparameters
                DataAugmentation: 'none'
                   Normalization: 'zerocenter'
            NormalizationDimension: 'auto'
                            Mean: [224×224×3 single]
```

```
inputSize = net1.Layers(1).InputSize
```

```
     inputSize = 1×3

   224   224     3
```

Replace Final Layers

```
lgraph1 = layerGraph(net1);
[learnableLayer,classLayer] = findLayersToReplace(lgraph1);
[learnableLayer,classLayer]
```

```
     ans =

       1×2 Layer array with layers:

          1   'fc1000'                        Fully Connected
1000 fully connected layer
```

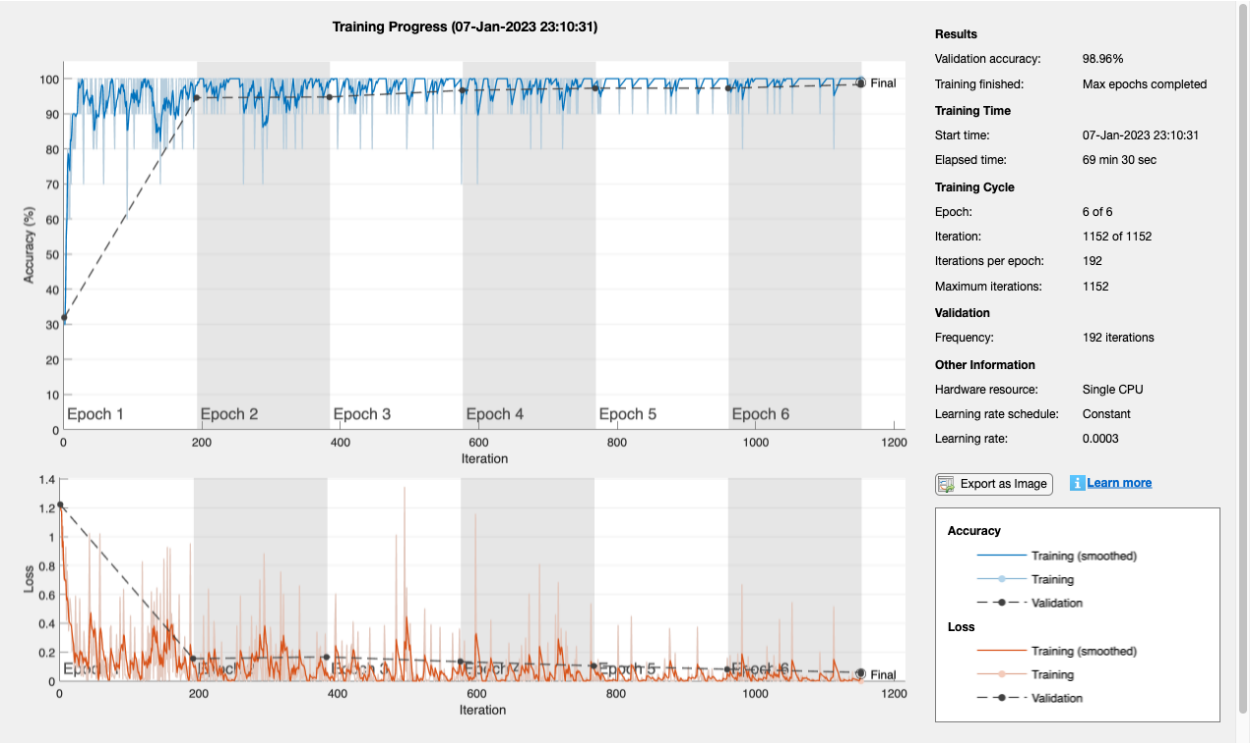
```
          2   'ClassificationLayer_fc1000'   Classification
Output   crossentropyex with 'tench' and 999 other classes
```

```
 if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
     newLearnableLayer = fullyConnectedLayer(numClasses, ...
         'Name','new_fc', ...
         'WeightLearnRateFactor',10, ...
         'BiasLearnRateFactor',10);

 elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
     newLearnableLayer = convolution2dLayer(1,numClasses, ...
         'Name','new_conv', ...
         'WeightLearnRateFactor',10, ...
         'BiasLearnRateFactor',10);
 end
 lgraph1 =
replaceLayer(lgraph1,learnableLayer.Name,newLearnableLayer);
 newClassLayer = classificationLayer('Name','new_classoutput');
 lgraph1 = replaceLayer(lgraph1,classLayer.Name,newClassLayer);
 layers = lgraph1.Layers;
 connections = lgraph1.Connections;
 layers(1:10) = freezeWeights(layers(1:10));
 lgraph1 = createLgraphUsingConnections(layers,connections);
```

Train Network

```
 pixelRange = [-30 30];
 scaleRange = [0.9 1.1];
 imageAugmenter = imageDataAugmenter( ...
     'RandXReflection',true, ...
     'RandXTranslation',pixelRange, ...
     'RandYTranslation',pixelRange, ...
     'RandXScale',scaleRange, ...
     'RandYScale',scaleRange);
 augimdsTrain = augmentedImageDatastore(inputSize,imdsTrain, ...
     'DataAugmentation',
imageAugmenter,'ColorPreprocessing','gray2rgb');
```

```
 augimdsValidation =
augmentedImageDatastore(inputSize,imdsValidation,"ColorPreprocessing",
"gray2rgb");
 miniBatchSize = 10;
 valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);
 options = trainingOptions('sgdm', ...
     'MiniBatchSize',miniBatchSize, ...
     'MaxEpochs',6, ...
     'InitialLearnRate',3e-4, ...
     'Shuffle','every-epoch', ...
     'ValidationData',augimdsValidation, ...
     'ValidationFrequency',valFrequency, ...
     'Verbose',false, ...
     'Plots','training-progress');
 classNames = net1.Layers(end).ClassNames;
 % train Renet50
 nettrain = trainNetwork(augimdsTrain,lgraph1,options);
```



Save network

```
save nettrain
```

Classify test set images

```
[YPrednet1,scoresnet1] = classify(nettrain,augimdsValidation);
```

Display four sample validation images and their labels

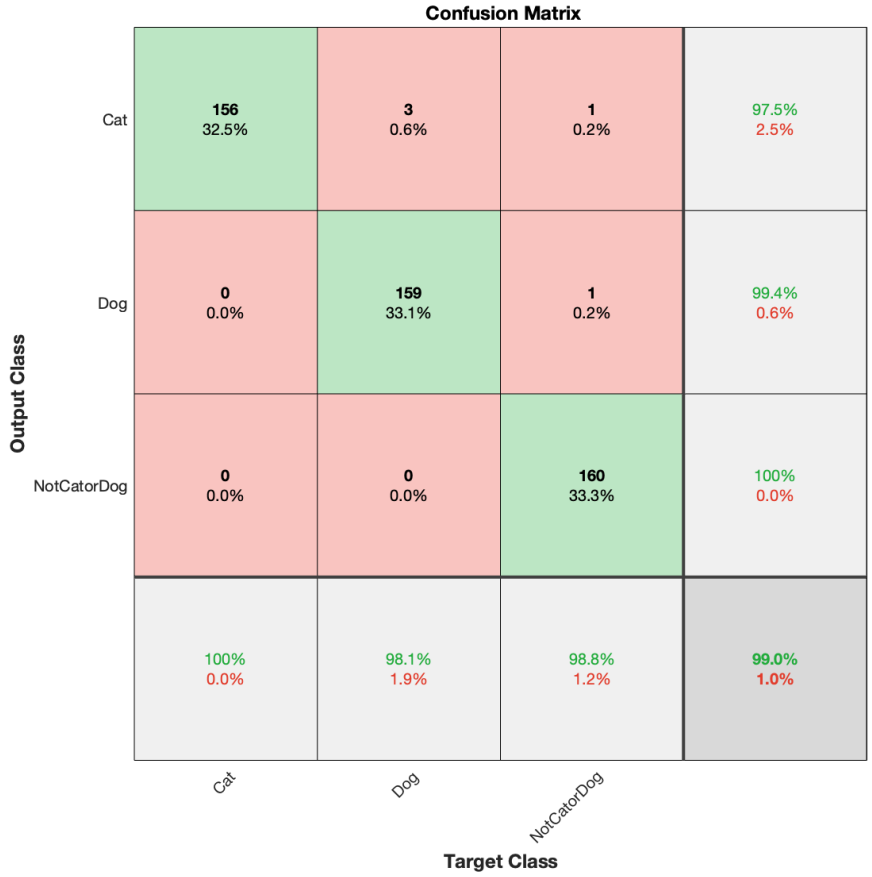
```
r = randperm(augimdsValidation.NumObservations,16);
for i = 1:numel(r)
    subplot(4,4,i)
    I = readimage(imdsValidation,r(i));
    label1 = YPrednet1(r(i));
    imshow(I)
    title(label1)
end
```

Calculate accuracy

```
YValidationnet1= imdsValidation.Labels;
accuracy1 = mean(YPrednet1 == YValidationnet1)
```

```
accuracy1 = 0.9896
```

```
plotconfusion(YPrednet1,YValidationnet1)
```



Googlelanet

```
net2=googlenet;
net2.Layers(1)
```

```
ans =

  ImageInputLayer with properties:


                       Name: 'data'

                  InputSize: [224 224 3]

          SplitComplexInputs: 0
```

```
        Hyperparameters
               DataAugmentation: 'none'
                  Normalization: 'zerocenter'
         NormalizationDimension: 'auto'
                           Mean: [224×224×3 single]
```

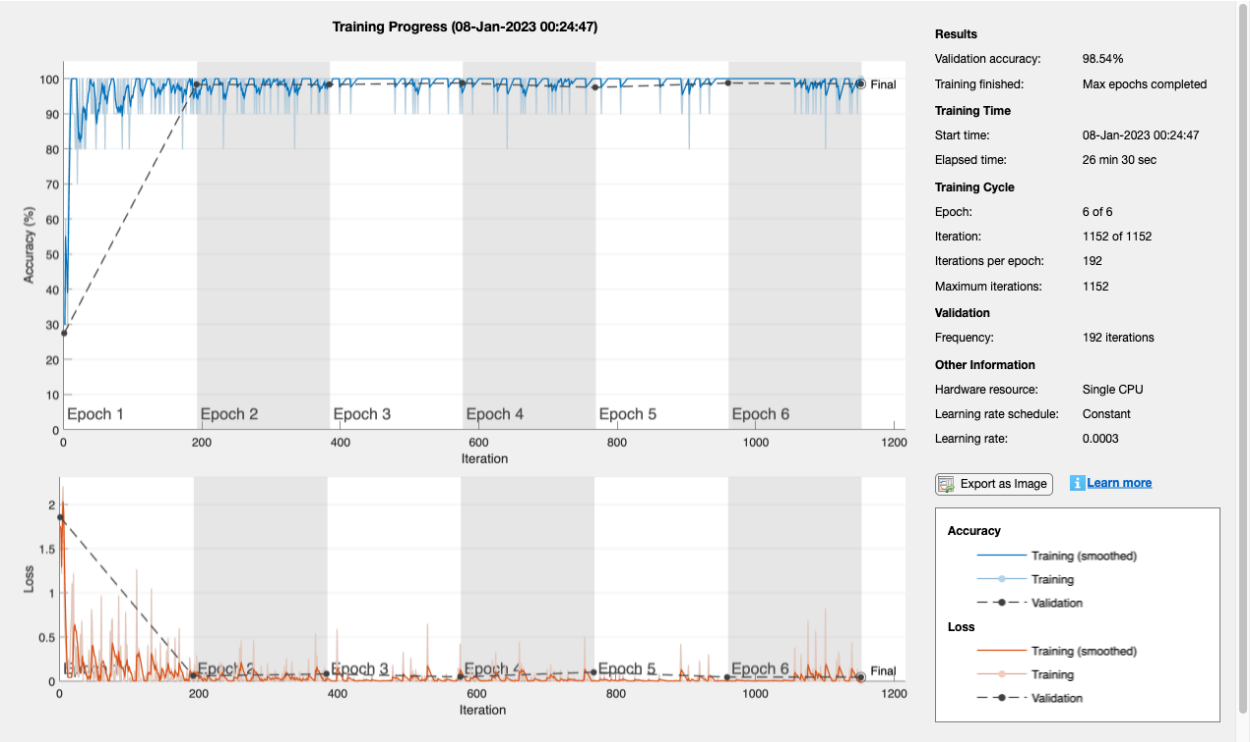
Replace Final Layers

```
lgraph2 = layerGraph(net2);
[learnableLayer,classLayer] = findLayersToReplace(lgraph2);
[learnableLayer,classLayer]
```

```
    ans =

      1×2 Layer array with layers:


         1   'loss3-classifier'   Fully Connected         1000
fully connected layer
         2   'output'             Classification Output
crossentropyex with 'tench' and 999 other classes
```

```
if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
```

```
        'BiasLearnRateFactor',10);
 end
 lgraph2 =
replaceLayer(lgraph2,learnableLayer.Name,newLearnableLayer);
 newClassLayer = classificationLayer('Name','new_classoutput');
 lgraph2 = replaceLayer(lgraph2,classLayer.Name,newClassLayer);
 layers2 = lgraph2.Layers;
 connections = lgraph2.Connections;
 layers2(1:10) = freezeWeights(layers2(1:10));
 lgraph2 = createLgraphUsingConnections(layers2,connections);
 nettrain2 = trainNetwork(augimdsTrain,lgraph2,options);
```



Save network

```
 save nettrain2
 [YPrednet2,scoresnet2] = classify(nettrain2,augimdsValidation);
```

Display four sample validation images and their labels

```
 r = randperm(augimdsValidation.NumObservations,16);
 for i = 1:numel(r)
     subplot(4,4,i)
     I = readimage(imdsValidation,r(i));
     label2 = YPrednet2(r(i));
```

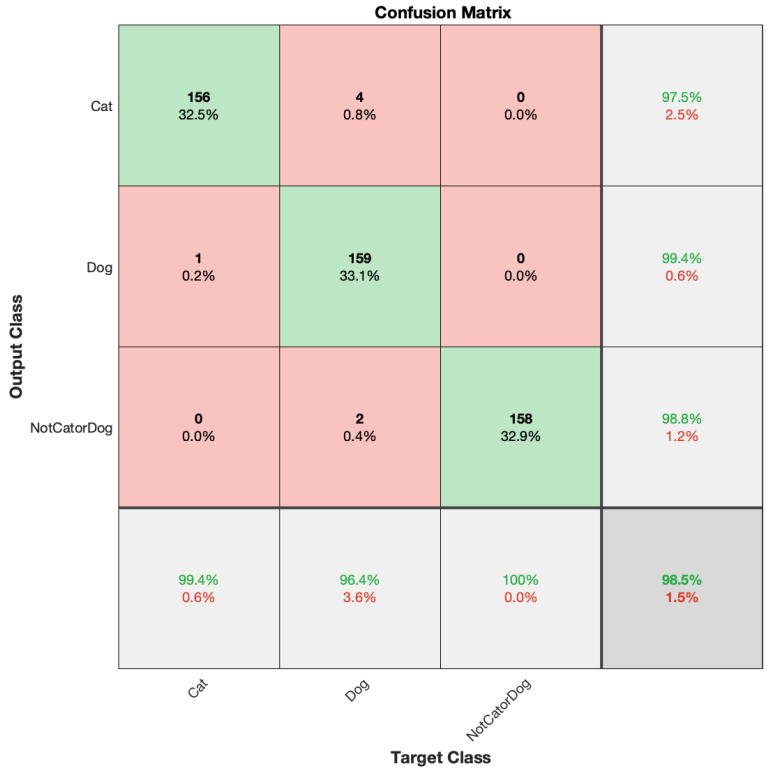
```
    imshow(I)
    title(label2)
end
```

Calculate accuracy

```
YValidationnet2= imdsValidation.Labels;
accuracy2 = mean(YPrednet2 == YValidationnet2)
```

```
accuracy2 = 0.9854
```

```
plotconfusion(YPrednet2,YValidationnet2)
```



Alexnet

```
net3=alexnet;
net3.Layers(1)
```

```
ans =

  ImageInputLayer with properties:

                      Name: 'data'
                 InputSize: [227 227 3]
       SplitComplexInputs: 0

   Hyperparameters
          DataAugmentation: 'none'
```

```
                    Normalization: 'zerocenter'
          NormalizationDimension: 'auto'
                             Mean: [227×227×3 single]
```

```
% resize image
inputSize2 = net3.Layers(1).InputSize
```

```
    inputSize2 = 1×3

   227   227     3
```

Replace Final Layers

```
lgraph3 = layerGraph(net3);
[learnableLayer,classLayer] = findLayersToReplace(lgraph3);
[learnableLayer,classLayer]
```

```
    ans =

      1×2 Layer array with layers:


         1   'fc8'      Fully Connected         1000 fully
connected layer
         2   'output'   Classification Output   crossentropyex
with 'tench' and 999 other classes
```

```
if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
```

```
        'BiasLearnRateFactor',10);
 end
 lgraph3 =
replaceLayer(lgraph3,learnableLayer.Name,newLearnableLayer);
 newClassLayer = classificationLayer('Name','new_classoutput');
 lgraph3 = replaceLayer(lgraph3,classLayer.Name,newClassLayer);
 layers3 = lgraph3.Layers;
 connections = lgraph3.Connections;
 layers3(1:10) = freezeWeights(layers3(1:10));
 lgraph3 = createLgraphUsingConnections(layers3,connections);

 pixelRange = [-30 30];
 scaleRange = [0.9 1.1];
 imageAugmenter = imageDataAugmenter( ...
     'RandXReflection',true, ...
     'RandXTranslation',pixelRange, ...
     'RandYTranslation',pixelRange, ...
     'RandXScale',scaleRange, ...
     'RandYScale',scaleRange);
 augimdsTrain2 = augmentedImageDatastore(inputSize2,imdsTrain, ...
     'DataAugmentation',
imageAugmenter,'ColorPreprocessing','gray2rgb');
 augimdsValidation2 =
augmentedImageDatastore(inputSize2,imdsValidation,"ColorPreprocessing"
,"gray2rgb");
 miniBatchSize = 10;
 valFrequency = floor(numel(augimdsTrain2.Files)/miniBatchSize);
 options = trainingOptions('sgdm', ...
     'MiniBatchSize',miniBatchSize, ...
     'MaxEpochs',6, ...
     'InitialLearnRate',3e-4, ...
     'Shuffle','every-epoch', ...
     'ValidationData',augimdsValidation2, ...
     'ValidationFrequency',valFrequency, ...
     'Verbose',false, ...
     'Plots','training-progress');
 % train alexnet
 nettrain3 = trainNetwork(augimdsTrain2,lgraph3,options);
```

Save network

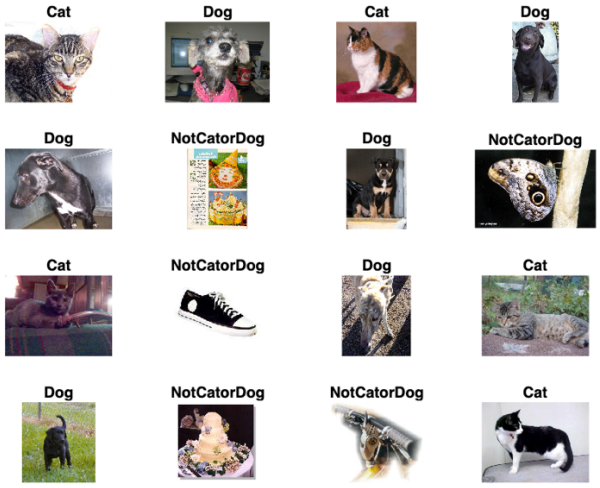
```
save nettrain3
```

Classify test set images

```
[YPrednet3,scoresnet3] = classify(nettrain3,augimdsValidation2);
```

Display four sample validation images and their labels

```
r = randperm(augimdsValidation2.NumObservations,16);
for i = 1:numel(r)
    subplot(4,4,i)
    I = readimage(imdsValidation,r(i));
    label3 = YPrednet3(r(i));
    imshow(I)
    title(label3)
end
```

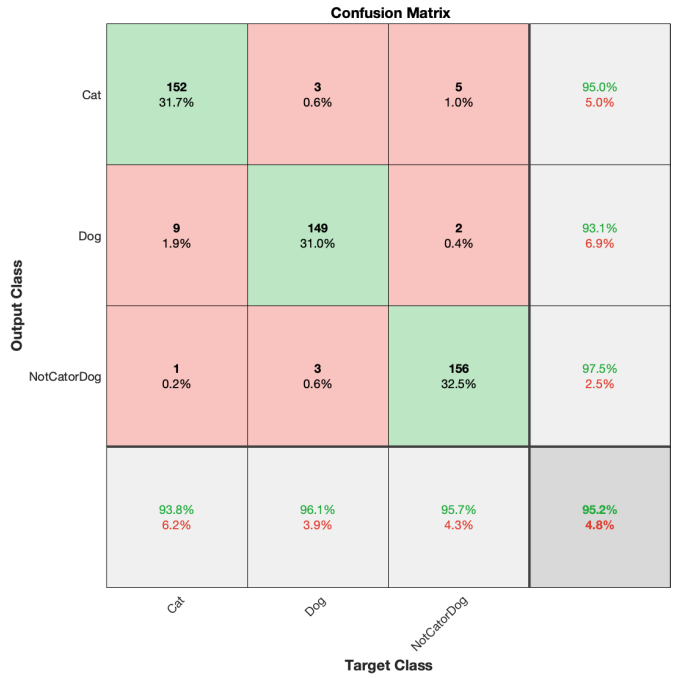


Calculate accuracy

```
YValidationnet3= imdsValidation.Labels;
accuracy3 = mean(YPrednet3 == YValidationnet3)
```

```
accuracy3 = 0.9521
```

```
plotconfusion(YPrednet3,YValidationnet3)
```



private function

```
function[learnableLayer,classLayer] = findLayersToReplace(lgraph)
if ~isa(lgraph,'nnet.cnn.LayerGraph')
    error('Argument must be a LayerGraph object.')
end
% Get source, destination, and layer names.
src = string(lgraph.Connections.Source);
dst = string(lgraph.Connections.Destination);
layerNames = string({lgraph.Layers.Name}');

% Find the classification layer. The layer graph must have a single
% classification layer.
isClassificationLayer = arrayfun(@(l) ...

(isa(l,'nnet.cnn.layer.ClassificationOutputLayer')|isa(l,'nnet.layer.C
lassificationLayer')), ...
    lgraph.Layers);

if sum(isClassificationLayer) ~= 1
    error('Layer graph must have a single classification layer.')
end
classLayer = lgraph.Layers(isClassificationLayer);
```

```
 % Traverse the layer graph in reverse starting from the
classification
 % layer. If the network branches, throw an error.
 currentLayerIdx = find(isClassificationLayer);
 while true

     if numel(currentLayerIdx) ~= 1
         error('Layer graph must have a single learnable layer
preceding the classification layer.')
     end

     currentLayerType = class(lgraph.Layers(currentLayerIdx));
     isLearnableLayer = ismember(currentLayerType, ...

['nnet.cnn.layer.FullyConnectedLayer','nnet.cnn.layer.Convolution2DLay
er']);

     if isLearnableLayer
         learnableLayer =  lgraph.Layers(currentLayerIdx);
         return
     end

     currentDstIdx = find(layerNames(currentLayerIdx) == dst);
     currentLayerIdx = find(src(currentDstIdx) == layerNames);
 end
 end

 function layers = freezeWeights(layers)
 % layers = freezeWeights(layers) sets the learning rates of all the
 % parameters of the layers in the layer array |layers| to zero.

 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 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
```