CAP 4630 – Intro to AI – Dr. Oge Marques

Final Project: Cats vs. Dogs

In this project you will design and implement a machine learning solution for image classification based on images of cats and dogs.

The goal is to build a machine learning solution in MATLAB that is capable of predicting whether an input color image containing an animal represents a *cat* or a *dog*.

You can choose whatever preprocessing image processing techniques, machine learning algorithms, and MATLAB toolboxes you wish, but keep in mind that the most important points are:

- To demonstrate the use of a sound **methodology** for every step of your work (e.g., feature selection, dataset selection, cross-validation techniques, performance measures, etc.).
- To **document** each step (and associated design decisions, criteria, thresholds, references, algorithms, strengths and weaknesses) of your work in a clear and readable manner.

The expected deliverables are:

- Self-contained MATLAB code to run your solution (and details on how to install and run it)
- A descriptive report containing the most important steps, building blocks, design decisions, examples of incorrect prediction results, and a ***summary table*** that should contain overall accuracy rates for, **at least, two different machine learning algorithms (and their variants, if applicable).**

Goals:

- Learn how to implement a complete, fully functional, machine learning solution for a contemporary computer vision challenge using MATLAB
- Get acquainted with representative contemporary datasets, challenges and problems in computer vision and machine learning
- Learn how to use deep neural networks under the paradigm of *transfer learning*
- Demonstrate the ability to perform feature selection, model selection, fine-tuning, and comparison, and performance evaluation of different solutions to the same problem

Starter package

- **Dataset**: cats and dogs dataset from Kaggle
  (https://www.kaggle.com/c/dogs-vs-cats/data)

- **Reference links**:
  - https://www.kaggle.com/c/dogs-vs-cats (more details about the Kaggle challenge associated with the dataset)
  - (OPTIONAL) Dogs vs. Cats Redux: Kernels Edition (a playground offered by Kaggle, if you want to learn more about Kernels, Jupyter Notebook, and

Final Project: Cats vs. Dogs

TensorFlow - https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition
- https://www.mathworks.com/help/nnet/examples/transfer-learning-usingconvolutional-neural-networks.html (transfer learning using MATLAB)

- **MATLAB starter code**: ***final_project_starter_cap4630.m*** (and associated data) (available on Canvas)

Instructions:

- This is a group activity
- Open collaboration among groups is **not** allowed

Procedure:

1. Download MATLAB starter code (and associated data) and add to your working folder, adjusting the MATLAB path if necessary.

2. Download the **train.zip** data file from the dataset link indicated above. You can disregard test1.zip (for now) and the sampleSubmission CSV file (forever).

Data Files

File Name	Available Formats
sampleSubmission	.csv (86.82 kb)
test1	.zip (271.15 mb)
train	.zip (543.16 mb)

3. Run "Part 1" of the starter code and ensure that it works as intended. Inspect the contents of variable **convnet**.

```
 1   'input'                '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'                'Convolution'                  '256 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'                'Convolution'                  '384 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]'
13   'relu4'                'ReLU'                         'ReLU'
14   'conv5'                'Convolution'                  '256 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   'fc7'                  'Fully Connected'              '4096 fully connected layer'
20   'relu7'                'ReLU'                         'ReLU'
21   'fc8'                  'Fully Connected'              '1000 fully connected layer'
22   'prob'                 'Softmax'                      'softmax'
23   'classificationLayer'  'Classification Output'        'crossentropyex with 'n01440764', 'n01443537', and 998 other classes'
```

4. Run "Part 2" of the starter code and ensure that it works as intended. What type of preprocessing is performed by the auxiliary function *readAndPreprocessImage*?

The function coverts any grayscale images to RGB and also formats all images to a size of 227x227.

5. Run "Part 3" of the starter code and ensure that it works as intended. What can you say about the montage with network weights for the second convolutional layer?

The montage of weights shows very primitive features that show edges and blobs. These features will become richer and better suited for recognition at deeper levels of the CNN.

6. Run "Part 4" of the starter code and ensure that it works as intended.
What other MATLAB function could have been used to build the SVM classifier in this case (binary classification problem)?

We could have used the fitcsvm function to train or cross-validate, using the ClasssificationSVM Class.

7. Run "Part 5" of the starter code and ensure that it works as intended.
Did your classifier recognize 'Doge' as a dog? If not, can you tell why?

The classifier was unable to recognize the Doge image because our CNN has only been trained on 20 pictures of dogs and cats. It will need more data to be accurate.

8. The starter code used a very small subset of the Kaggle dataset (20 images of dogs and 20 images of cats) out of the 25,000 images (2 x 12,500) available. This has to be changed.

In this step, you should partition your dataset into:

- Training: TR %
- Cross-validation: CV %
- Testing: TS %

Recommended values for TR, CV, and TS are 60, 20, and 20, respectively. If you use different values (or "play" with different choices), make sure to document that.

9. (OPTIONAL) Perform additional image preprocessing (e.g., sharpening, brightness/contrast/color adjustments, etc.) on all images from the dataset.

10. Select a (family of) classifier(s) to use **first**. Recommended techniques include (but are not limited to):

a. Logistic Regression
b. SVM (linear, quadratic, cubic, RBF Gaussian kernel, etc.)
c. K nearest neighbors (kNN)
d. Naïve Bayes
e. Decision Tree
f. Discriminant Analysis
g. Neural networks
h. Deep neural networks

11. Implement the entire workflow to test your solution and produce meaningful figures of merit and plots (see Warm-up Exercise).

12. (OPTIONAL) Tweak and fine-tune meaningful parameters associated with your first solution. Then freeze it and add meaningful results to the "summary table".

13. Select a different (family of) classifier(s) to use **second**.

14. Implement the entire workflow to test your solution and produce meaningful figures of merit and plots (see Warm-up Exercise).

15. (OPTIONAL) Tweak and fine-tune meaningful parameters associated with your second solution. Then freeze it and add meaningful results to the "summary table".

16. (OPTIONAL) Perform, test, and document any relevant changes or improvements (additional classifiers, different metrics, different thresholds, different partitioning of the dataset, etc.)

17. Produce a meaningful technical report. In addition to the summary table include, if applicable, other tables and plots, as well as representative samples of best and worst results.

Deliverables

You should submit **a single zip file** via Canvas.

It should contain:

- All necessary m-files to properly reproduce your results.
- All relevant third-party MATLAB scripts and functions that were ***not*** provided in the starter package (with an acknowledgment of their source), if any.
- A ***detailed*** report (PDF) containing your comments, remarks, examples, answers and comments associated with the Warm-up Exercise, relevant tables and plots, etc.
- A README file describing what is in the package, and (optionally) containing instructions on how to “install” (e.g., directory structure, dependencies, etc.) and run your code.

It should ***not*** contain:

- Any images or files that were already distributed as part of the starter package.

- The actual dataset.
- Older versions of your code or any leftovers that are not relevant for (and might even hurt) grading your work.

Grading rubric

Code: quality	10%
Code: correctness	30%
Code: inline documentation	5%
README file	5%
Report	50%

REPORT

Alexnet

My code is implemented on top of Prof. Marques' starter code. My section of code, importing the Kaggle data set and setting up/ training my alexnet model, is implemented in Part 6 of the ALEXNET.m code.

In 6.1, we load the new training set as an image datastore object and import our 'cat' and 'dog' labels. Once we have formatted our images to the 227x227 crop, we can then go ahead and partition our dataset into its respective sections for learning. In our alexnet model we used 80% of our data set to train, then 20% for validation and the other 20% for testing and accuracy of the model.

```
[trainingImages,validationImages, testingImages] = splitEachLabel(images,0.6, 0.2,'randomized');
```

To make sure our images were formatted correctly, we implement the plot of images 'for loop' that was featured in the starter code package.

In 6.2, we focus on setting up our alexnet model for training on the new dataset. Firstly, we extrapolate all but the last 3 fully connected layers of our cnn with:

```
layersTransfer = convnet.Layers(1:end-3);
```

Then we add in our new connected layers by redefining our layers variable as:

```
layers = [
layersTransfer
fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20, 'BiasLearnRateFactor',20)
softmaxLayer
classificationLayer];
```

In 6.3, we finally set up our training options and run our model. We set our mini batch size equal to 10, and then defined our number of iterations per epoch to:

```
numIterationsPerEpoch = floor(numel(trainingImages.Labels)/miniBatchSize);
```

Once set, we then set our training options using the trainingOptions function from the Neural Networks Toolbox. For our model, we stuck with preselected options which Mathworks gives when giving their example implementation of alexnet:

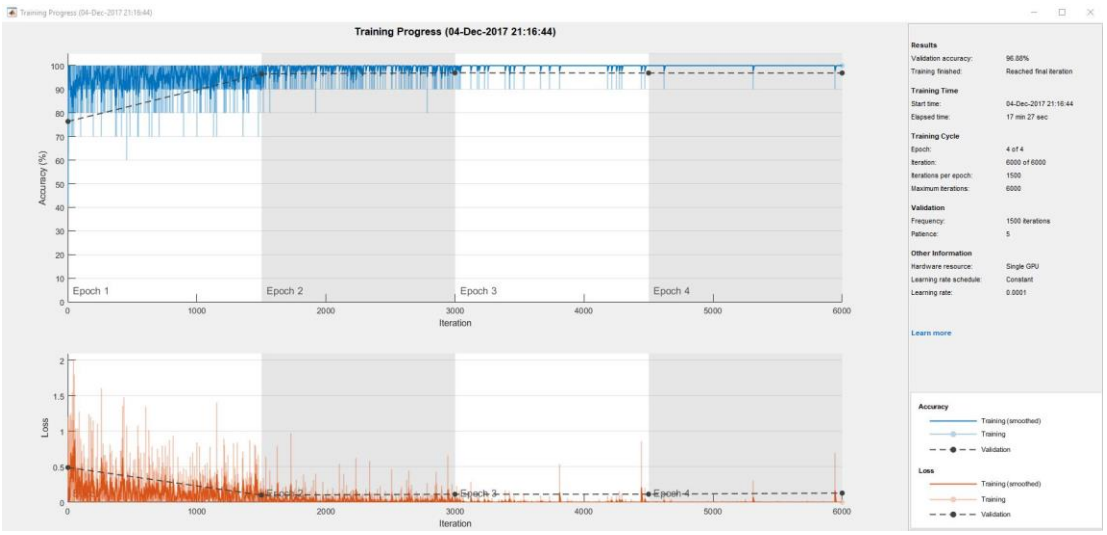
```
options = trainingOptions('sgdm',...
        'MiniBatchSize',miniBatchSize,...
        'MaxEpochs',4,...
        'InitialLearnRate',1e-4,...
        'Verbose',false,...
        'Plots','training-progress',...
        'ValidationData',validationImages,...
        'ValidationFrequency',numIterationsPerEpoch);
```

It should be noted here that our model was trained using the newest version of Matlab (R2017b) and the Neural Network Toolbox has changed since earlier versions, so it is recommended to implement this code on the version of Matlab we used to ensure no compatibility issues occur.

From here we simply need to type out trainNetwork function, also easily accessible thanks to our Neural Networks toolbox.

```
netTransfer = trainNetwork(trainingImages,layers,options);
```

Once the model loads, a plot should appear and start to show training progress:



Our model was able to train and validate with an accuracy of 96.88%. training took 17min. 27sec. with a GTX 960 6gb GPU. When testing our model, the accuracy came out similar but higher, at 98.8%.

VGG-19 CNN

For our second classifier, we decided to implement a well-known convolutional neural network called VGG-19. Located in the VGG19.m matlab code file. In part 1, we first download and inspect the pre-trained network and take a look at the layers that make up the network.

```
vgg19;
convnet = vgg19;
convnet.Layers
```

On closer inspection, we are able to see the 19 layer CNN's full matlab implementation:

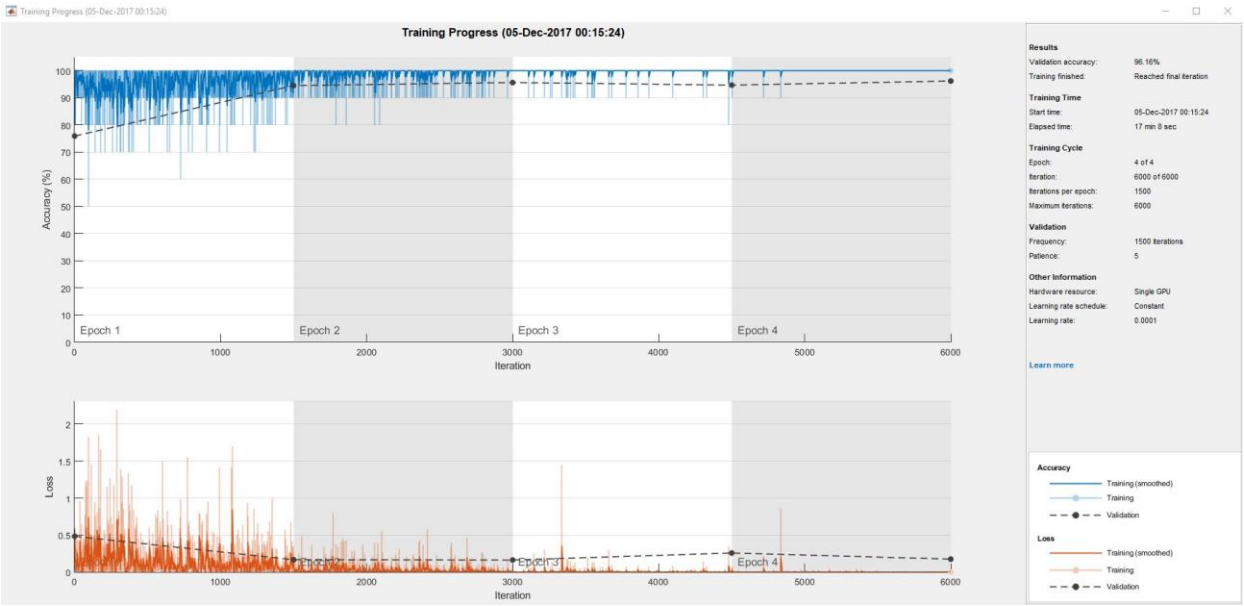
```
ans =

  47x1 Layer array with layers:

     1   'input'     Image Input             224x224x3 images with 'zerocenter' normalization
     2   'conv1_1'   Convolution             64 3x3x3 convolutions with stride [1  1] and padding [1  1]
     3   'relu1_1'   ReLU                    ReLU
     4   'conv1_2'   Convolution             64 3x3x64 convolutions with stride [1  1] and padding [1  1]
     5   'relu1_2'   ReLU                    ReLU
     6   'pool1'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
     7   'conv2_1'   Convolution             128 3x3x64 convolutions with stride [1  1] and padding [1  1]
     8   'relu2_1'   ReLU                    ReLU
     9   'conv2_2'   Convolution             128 3x3x128 convolutions with stride [1  1] and padding [1  1]
    10   'relu2_2'   ReLU                    ReLU
    11   'pool2'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
    12   'conv3_1'   Convolution             256 3x3x128 convolutions with stride [1  1] and padding [1  1]
    13   'relu3_1'   ReLU                    ReLU
    14   'conv3_2'   Convolution             256 3x3x256 convolutions with stride [1  1] and padding [1  1]
    15   'relu3_2'   ReLU                    ReLU
    16   'conv3_3'   Convolution             256 3x3x256 convolutions with stride [1  1] and padding [1  1]
    17   'relu3_3'   ReLU                    ReLU
    18   'conv3_4'   Convolution             256 3x3x256 convolutions with stride [1  1] and padding [1  1]
    19   'relu3_4'   ReLU                    ReLU
    20   'pool3'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
    21   'conv4_1'   Convolution             512 3x3x256 convolutions with stride [1  1] and padding [1  1]
    22   'relu4_1'   ReLU                    ReLU
    23   'conv4_2'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    24   'relu4_2'   ReLU                    ReLU
    25   'conv4_3'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    26   'relu4_3'   ReLU                    ReLU
    27   'conv4_4'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    28   'relu4_4'   ReLU                    ReLU
    29   'pool4'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
    30   'conv5_1'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    31   'relu5_1'   ReLU                    ReLU
    32   'conv5_2'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    33   'relu5_2'   ReLU                    ReLU
    34   'conv5_3'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    35   'relu5_3'   ReLU                    ReLU
    36   'conv5_4'   Convolution             512 3x3x512 convolutions with stride [1  1] and padding [1  1]
    37   'relu5_4'   ReLU                    ReLU
    38   'pool5'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
    39   'fc6'       Fully Connected         4096 fully connected layer
    40   'relu6'     ReLU                    ReLU
    41   'drop6'     Dropout                 50% dropout
    42   'fc7'       Fully Connected         4096 fully connected layer
    43   'relu7'     ReLU                    ReLU
    44   'drop7'     Dropout                 50% dropout
    45   'fc8'       Fully Connected         1000 fully connected layer
    46   'prob'      Softmax                 softmax
    47   'output'    Classification Output   crossentropyex with 'tench', 'goldfish', and 998 other classes
```

In Part 2 of our code, we implement a very similar technique used for importing our data set for alexnet, as well as for extrapolating and reconnecting our last three layers of the net. Because VGG-19's last three layers are all fully connected similar to alexnet, it was very easy to add our layers into the extrapolated layersTransfer neural network object.

From here, we simply set up our training options for our model and train our VGG net.



Our Model was able to complete its 4 epochs in 17min. and 8sec. on the same GTX 960. Our Validation Accuracy was 96.16%, while our testing accuracy on our partitioned dataset came out to 98.9%, slightly better than our trained alexnet.

Conclusion

Things that we could do moving forward from this project formally, would be to download the testing set from Kaggle and submit a .csv file to see how well we would have placed in the leaderboard. Personally, going forward, I know I will be constantly looking at the competitions hosted on Kaggle, not only are they a good source for information on how others are implementing their models, but they are a great place to practice and to discover new datasets to work with.

This project truly helped me understand the nuances of how neural networks and transfer learning specifically can be implemented during to a classification problem.