##################################

# MATLAB: DeepLearnToolbox

###################################

HOW TO RUN CODE

-----NEURAL NETWORK CODE: 'Coletti-DeepLearnToolbox/NN/'-----

\*\*\*This directory contains the main feed-forward neural network code from the original toolbox. We modified many of the functions and setup files to allow for our desired training hyperparameters, corruption types, and intermediate save settings.

**nnadjust.m**

A function written, by us, to adjust the network weights during testing so that it would accurately predict examples with just one forward pass. It was inspired by the same logic that Hinton uses to cut the weights of his dropout network in half when testing. This adjustment method did not perform well in practice, but represented an important experiment.

**nnbp.m**

Applies backpropagation to the network. We modified this code so that it does NOT propagate error back through corrupted hidden units. Because the output of the unit after corruption has no relationship to the output of the unit before corruption, we do not want to unfairly penalize hidden units that may have been predicting accurately.

**nnff.m**

The feed-forward pass of the neural network. We modified this file to allow for different types of corruption other than standard dropout. It randomly selects the mask, and then applies the correct type of corruption.

**nntrain.m**

We modified the high-level training file to write the error rates to file rather than just printing them. Furthermore, we added an intermediate save point every X epochs. We made the file name encode all the identifying information of the network, so multiple networks could be trained at the same time w/out overwriting eachother.

Additionally, we modified the training file to corrupt the inputs with the desired corruption type

**nntrain\_connect.m**

This is a variation of nntrain.m which was modified to allow for the fixed training schedule of the dropConnect paper. We used this training schedule for the milestone.

**nnsetup.m**

The configuration file for the network (similar to the prototxt from caffe). We modified this file to include the hyperparameters for the training schedule and additional parameters necessary for the other types of corruption.

-----TESTING CODE: 'Coletti-DeepLearnToolbox/tests/'-----

**test\_hinton.m**

Function that trains, stores, and evaluates a feed-forward network of any corruption type using the training schedule from Hinton et al. (2010)

**test\_connect.m**

Function that trains, stores, and evaluates a feed-forward network of any corruption type using the training schedule from Wan et al. (2013)

-----ANALYSIS CODE: 'Coletti-DeepLearnToolbox/analysis/'-----

\*\*\*This directory contains code added by our group, not in the original DeepLearnToolbox

**get\_ensemble\_confusionMat.m**

Gets the mean confusion matrix and the standard deviation of each entry for an ensemble of networks given a directory that contains them.

**get\_nn\_confusionMat.m**

Gets the confusion matrix for a single network.

**get\_nn\_filters.m**

Creates an image of 64 filters from the first hidden layer.

**get\_nn\_mapping\_vector.m**

Creates a vector by concatenating the softmax outputs of a network for each of 10 examples in every digit class to be used as a representation of its input-output mapping.

**get\_nns\_mapping\_matrix.m**

Compiles a matrix of mapping vectors by looping through networks in a given directory and calling get\_nn\_mapping\_vector().

**graph\_nn\_corruption.m**

Graphs the 2D tSNE visualization of networks of different corruption types given a list of directories that are used to call get\_nns\_mapping\_matrix().

**plot\_nn\_corrupts\_err.m**

Plots test error over time for networks of different corruption types.

**plot\_nn\_err.m**

Plots training and test error over time for a single network.

**visualize\_confusionMat.m**

Emphasizes parts of confusion matrix that were wrong by taking the log of a constant times its values and resizing the image to five times its original scale.

**visualize\_nn\_weights.m**

Attempts to visualize weights of upper layers (as well as the first layer by calling get\_nn\_filters()) by taking the average of images in the training set weighted by the activations of each hidden unit when that example is being fed forward through the network. However, the resulting filters are not sufficiently different to make a reliable comparison using this method.

**dir\_nnPredictN.m**

for every NN saved in a directory, this funciton will run nn\_findAvgPredictN and store the returned average softmax output as a file with the same name plus 'PREDICTIONS' appended at the end.

**nn\_findAvgPredictN.m**

Performs N feed-forward passes through the network and returns the average softmax output over all N passes. Averaged softmax output that is returned is an accurate prediction, allowing the network to be properly evaluated using the corruption from training.

**get\_EnsembleError{Max,Median,Min,Product}.m**

These functions search through the results directory, finding all of the "prediction" files of each corruption type and corruption rate. It performs ensemble prediction of the corresponding method for each of these combinations and returns in a cell array the ensemble performance across all corruption types (dim 1), corruption rates (dim 2), and ensemble sizes (the index into the vector stored at the corresponding corruption type and corruption rate).

**plot\_ensembleError.m**

This script creates plots that show the effect of corruption type and corruption rate on ensemble prediction performance. It creates a plot for each corruption type, with each corruption rate having its own curve. It also creates a plot for each corruption rate, with each corruption type having its own curve. It uses mean polling to assess ensemble prediction error.

**plot\_ensembleMethods.m**

This script creates plots that show the effect of different ensemble prediction methods on test error. It creates a plot of ensemble size against test error for each corruption type and corruption rate

**plot\_predictErrorAvg.m**

This function plots the prediction error of a network of each corruption type against the number of feed-forward passes averaged to visualize the convergence of the prediction error as large amounts of feed-forward passes are performed.

##################################

# Caffe

###################################

\*\*\* We did not actually end up using any of the models from Caffe in our final project. However, the effort put getting Caffe working, digging through the source code to understand the models, and making slight adjustments that would allow for us to use it for Cifar-10 represented a large enough amount of time (> 20 hours) to warrant inclusion.

To use train networks using caffe code, follow the tutorials at http://caffe.berkeleyvision.org/ which outline how to use the examples directory. After training networks, use you can extract features by running the extract\_features shell scripts in the respective examples directories.

**'caffe/tools/extract\_features\_csv.cpp'**

This function was modified to write the features to csv instead of to a level-db. The 'blob' datastructure was extremely complex to understand, as it is almost entirely abstracted away by the Google protobuf serialization structure. We intended to use this to extract features from a trained network on Cifar10 so we could then train a single fully-connected layer on top with our Matlab code.

**'caffe/tools/protobuf\_to\_csv.cpp'**

This function was created to read a level-db and write it to file. This was another attempted hack to get the extracted features from the cifar10 network.

**'caffe/src/caffe/layers/accuracy\_layer.cpp'**

This file was modified to output the correct label to a file, for to use in training the network in a separate program (MATLAB code), as the example ordering was randomized

**'caffe/examples/mnist/csv\_extract\_features\_lenet.sh'**

shell script to run the extract\_features\_csv.cpp file on the trained lenet CNN.

**'caffe/examples/mnist/extract\_labels\_lenet.sh'**

shell script to run the test\_net.cpp file on the trained lenet network and output the labels to a file using the modified accuracy layer file.

**'caffe/examples/cifar10/'**

all of the .prototxt files in this directory were modified so that a the "quick" and "full" networks could be trained simultaneously. That meant creating two separate level-db directories with their own copies of the data and accessing them separately.

**'../caffe/examples/cifar10/csv\_extract\_features\_cifar.sh'**

shell script to run the extract\_features\_csv.cpp file on the trained cifar10 CNN and write the features to file.

**'../caffe/examples/cifar10/extract\_labels\_cifar.sh'**

shell script to run write the labels of examples to file using the modified accuracy\_layers.cpp file.