5

CONVNET Manual

This chapter is a user's manual for the CONVNET program, available as a free download from the author's website. The first section lists every menu option, along with a brief description of its purpose and the page number on which more details can be found if the short description is not sufficient.

Menu Options

File Menu

Read control file - Page 182

A standard text file is read. This file contains architectural specifications for the model (this is the only way to define architecture), and optionally may contain commands to read or create input images or train the model.

Read MNIST image

A standard MNIST-format file is read. The corresponding label file must be read after the image file is read. Only one MNIST image/label pair may be read. Other file reading options are disabled after an MNIST image/label pair is read. It is assumed that there will be ten classes; this is hard-coded into the program. However, the size of the images is not hard coded. It is read from the file. The product of the number of rows times the number of columns cannot exceed 2^16-1=65,535. This unfortunate limitation comes from a hardware property of current CUDA devices which would be difficult to work around.

Read MNIST labels

A standard MNIST-format label file is read. It is assumed that there are ten classes. The corresponding MNIST image file must be read before the label file is read.

Read CIFAR-10 image

A standard CIFAR-10-format file is read. Multiple CIFAR-10 files may be read, in which case they are concatenated. This command cannot be used if MNIST or series data is already present.

Read series - Page 183

A univariate time series is read and a set of predictors is computed based on the values of the series, optionally differenced and/or log transformed. Class identities are generated. This selection brings up a menu in which parameters relevant to reading the series may be entered. These parameters, in the context of control files, are discussed starting on Page 183.

Make image

An artificial image having random tones is generated to enable quick and easy testing of data and model configurations. The user specifies the height and width, the number of bands, the number of classes, and the number of cases. This command cannot be used if a dataset is already present.

Clear all data

All training data is erased, but a trained model (if it exists) is retained. The purpose of this command is to allow reading a test dataset and evaluating the performance of a trained model on this new dataset. A common sequence of operations is ***Read training data, Train, Clear, Read test data, Test***.

Print

The currently selected display window (created under the *Display menu*) is printed. If no window is selected, *Print* is disabled.

Exit

The program is terminated.

Test Menu

Use CUDA (Toggle Yes/No)

This option is enabled only if a CUDA-capable device is present on the computer. If a check mark appears next to this option, the CUDA device will be used for compute-intensive operations. Click this option to toggle the check mark on and off.

Training params - Page 188

Parameters relevant to training can be set. This selection brings up a dialog box in which these parameters may be changed from their default values. The nature of these parameters is discussed in the context of a control file on Page 188.

Train - Page 191

The model is trained using the data currently present. It is important to understand which phases of training can and cannot be interrupted with the ESCape key. See Page 191 for details.

Test

The trained model is tested with the data currently present. The current version of CONVNET does not allow interruption of computing the confusion matrix; you'll just have to sit and wait for it to finish. Sorry. It's on my list, but for some technical reasons it's not a quick-and-easy fix. I hope to post updated versions of the program on my website as improvements occur.

Print model weights

All model weights are printed to the CONVNET.LOG file. This can be gigantic! Even modest models can have so many weights that writing them to the CONVNET.LOG file can take several minutes and consume megabytes. You've been warned.

Display Menu

Display training images - Page 192

A user-selectable set of the images in the current dataset is displayed.

Display filter images - Page 193

If a trained model exists, and the first hidden layer of this model is convolutional, this option displays as images the filter weights for a user-selectable set of slices.

Display activation images - Page 194

If a trained model exists, this option displays as images the activations of the visual field of the first hidden layer for a user-selectable set of slices and training case.

Read Control File

Intelligent readers will study this section and learn to perform most or all operations via a control file. Every CONVNET operation except specifying the model architecture can be done with the menu system, and that may be the preferable approach if one is just idly fooling around. However, in the vast majority of cases, it is best for the user to first create a control file using any ordinary text editor, and completely specify all project details in this file. This avoids tedious repetitive entry of parameters via the menu system, and it also provides hard documentation of all project specifications.

A control file is an ordinary text file. Each line of this file specifies a single aspect of the project. Comments can be inserted by starting a line with two forward slashes (//). This also provides a convenient mechanism for temporarily deactivating lines in the file without deleting them.

Making and Reading Image Data

This section describes methods for making random test images as well as reading popular-format image files.

MAKE IMAGE Rows Columns Bands Classes Cases

This produces a set of training images having random tones. The user specifies the height and width, the number of bands, the number of classes, and the number of cases. This command cannot be used if a dataset is already present.

READ MNIST IMAGE "FileName"

An MNIST image file is read. This command cannot be used if a dataset is already present. The corresponding label file must be read after the image file is read.

READ MNIST LABELS "FileName"

An MNIST label file is read. This command would normally follow a READ MNIST IMAGE command.

READ C10 IMAGE "FileName"

A CIFAR-10 image file is read. This command cannot be used if a dataset other than CIFAR-10 is already present. Multiple CIFAR-10 image files may be read, and their contents will be concatenated.

Reading a Time Series as Images

This is a powerful technique for converting a time series to a set of images. A moving window is passed across a time series. Each placement defines an image. This window image is divided into a user-specified number of rows (value of the series) and columns (relative time in the window). The path of the series is set to black in the image, and everything else is set to white. Figure 5.1 below shows a typical set of images produced from prices of OEX, the Standard and Poor's 100 index, as the window slides along left to right.

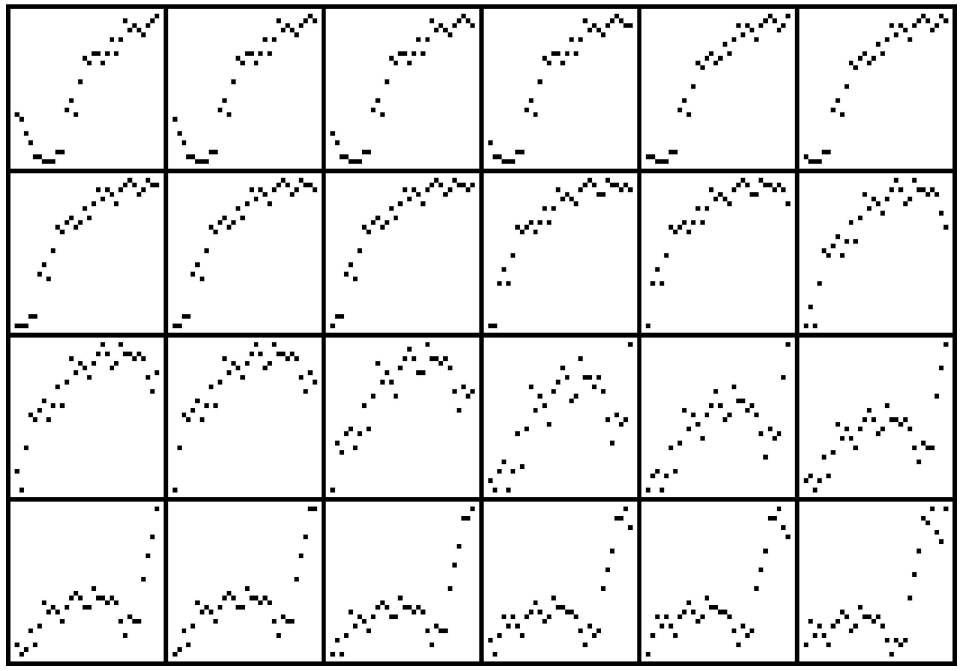

Figure 5.1: Series images from OEX

The command to read the series and produce the image set is shown below. Nothing else need be specified. However, in most cases the user will wish to change some specifications from their defaults. The legal specifications are also shown, with their default values indicated. Naturally, all such specifications must appear *before* the READ SERIES command to which they will apply.

The series file must be an ordinary text file. It may contain a header, and it may contain multiple columns. If there are multiple columns, spaces, tabs, and commas serve as delimiters. There is one observation per record.

READ SERIES "FileName"

A time series file is read. This command cannot be used if a dataset is already present. A moving window is applied to the series to produce a set of images.

SERIES COLUMN = Column

The series data can be fetched from any column. This specifies the column containing the desired values. The default is 1.

SERIES WINDOW = Width

This is the number of records in each window placement. Hence, it is the width of the images. The default is 16.

SERIES RESOLUTION = Resolution

This is the vertical resolution in each window placement. Hence, it is the height of the images. The default is 16.

SERIES SHIFT = Shift

This is the number of records that each window placement will advance to produce the next image. The default is 1.

SERIES RAWDATA

This, the default, specifies that the values read from the file are used as the series data.

SERIES RAWLOG

This specifies that the log of the values read from the file are used as the series data.

SERIES DIFFDATA

This specifies that the differences in the values read from the file are used as the series data. In other words, each computed series value is the current value of the file series minus the prior value.

SERIES DIFFLOG

This is identical to SERIES DIFFDATA except that the difference of the logs is used. Equivalently, this is the log of the ratios.

SERIES FRAC FULL = Fraction

This is the fraction (0-1) of training set cases that are forced to occupy the full vertical range of the window. Windows are not necessarily individually normalized (scaled), as this would distort information content. Normalization is usually relative to the entire series. A specification of zero maps the greatest range of the series across all windows to the full vertical range of the window, meaning that (except for ties) only one window will display the full vertical range. In many situations this will result in many or most windows having very little variation; they are essentially a flat line. A specification of one causes each window to be individually normalized, so all windows display the full vertical range. This is probably not good, as it fails to distinguish windows having little series variation from those having great variation; that's important information, and it's lost. The default is 0.2. This means that the 80'th percentile (1 minus 0.2) of within-window ranges is the variation that maps to the full vertical range for those 80 percent of cases. The 20 percent of windows whose series range exceeds this quantity are individually normalized to full vertical range. A simple way of thinking about this specification is that this is the fraction of cases that are individually normalized to the full vertical range. In most applications this should be well under 0.5.

SERIES TARGET NO DIFF

This, the default, specifies that the target class is determined by the next value in the series past the window. This determination will be based on the undifferenced or differenced nature of the series. In other words, the target will be determined by the difference between the next value outside the window minus the last value in the window, if and only if the user specifies that the series is differenced. *Differencing of the target matches the predictors.*

SERIES TARGET DIFF

Specify this option if the series is not differenced (RAWDATA or RAWLOG) but you want the target determination to be based on differences. This would be appropriate, for example, in financial market prediction.

SERIES CLASS ZERO

This, the default, specifies that the class of a case is defined by the sign of the target (which may or may not have been differenced, as above). One class is for targets greater than zero, and the other for targets less than or equal to zero.

SERIES CLASS MEDIAN

This specifies that the class of a case is defined by the value of the target relative to the median across the training set. One class is for targets greater than the median, and the other for targets less than or equal to the median.

SERIES CLASS THIRDS

This specifies that the class of a case is defined by the value of the target relative to the 33 and 66 percentiles across the training set. There are three classes, a low, middle, and high class.

SERIES NO HEADER

This, the default, specifies that the series file has no header record. The data begins with the first record.

SERIES HEADER

This specifies that the series file has a header, so the first record is skipped.

Model Architecture

The architecture of the model must be specified in a control file; there is no menu interface for doing so. Layers of the model are given in order from the first hidden layer to the last. There are no specifications for the input and output layers. The following layer types may be defined:

FULLY CONNECTED LAYER Slices

This creates a fully connected layer consisting of the specified number of slices. In architecture reports, it will appear as having one row, one column, and a depth equal to the number of slices.

LOCAL LAYER Slices hwV hwH padV padH strideV strideH

This creates a locally connected layer having the specified number of slices, vertical and horizontal half-widths, vertical and horizontal padding, and vertical and horizontal stride. The dimensions of the visual field of this layer are given by Equation (2.8) on Page 25.

CONVOLUTIONAL LAYER Slices hwV hwH padV padH strideV strideH

This creates a convolutional layer having the specified number of slices, vertical and horizontal half-widths, vertical and horizontal padding, and vertical and horizontal stride. The dimensions of the visual field of this layer are given by Equation (2.8) on Page 25.

POOLED AVERAGE LAYER widthV widthH strideV strideH

This creates an average pooling layer with the specified vertical and horizontal widths (not half-widths) and stride. The dimensions of the visual field of this layer are given by Equation (2.8) on Page 25. The number of slices is equal to the number in the prior layer.

POOLED MAX LAYER widthV widthH strideV strideH

This creates a max pooling layer with the specified vertical and horizontal widths (not half-widths) and stride. The dimensions of the visual field of this layer are given by Equation (2.8) on Page 25. The number of slices is equal to the number in the prior layer.

Training Parameters

The following parameters relevant to training may be set. Default values are as indicated. It may be that a revised CONVNET program may change these defaults from those that are printed here. The defaults for the current version of the program can be seen by selecting the *Test / Training parameters* menu option.

MAX BATCH = Number

This is relevant only for CUDA training. Kernel launches are divided into subsets of the full training set in order to prevent the infamous Windows WDDM timeout. This parameter limits the maximum number of cases in a subset. The default is 100. Lower this number to lower the per-launch time for all training steps.

MAX HID GRAD = Number

This is the maximum number of hidden neurons that will be processed per launch during CUDA gradient computation of convolutional and locally connected layers. Lowering this number can reduce the per-launch time for gradient computation, without affecting any other aspect of training. In many situations, it is best to leave this be huge and limit time with the next parameter, MAX MEM GRAD. The default is 65535, which is the maximum legal value.

MAX MEM GRAD = Number

This is the preferred way to lower the time required for gradient computation of convolutional and locally connected layers. It does not impact any other operations. This specifies the maximum memory in megabytes to dedicate to scratch work for convolutional hidden layers. A useful side effect is that limiting the memory causes launches to be broken into smaller sets of hidden neurons, which reduces the per-launch compute time and hence can prevent Windows WDDM timeouts. Lower this number to reduce per-launch compute time. You may also wish to use a smaller number if your CUDA device has limited on-board memory. The default is 2047 megabytes, which is the maximum legal value.

To summarize the prior three parameters.... Windows limits CUDA computation time for a single kernel launch. The limit is generally two seconds. If this time is exceeded, the screen will temporarily go black, and an error message will appear, and the application will be severely compromised. If this happens, you must reduce per-kernel time. Study the CUDA.LOG file to see where excessive per-launch time is occurring. Activation and gradient computation are the only serious time eaters. The MAX BATCH parameter impacts all operations. The MAX HID GRAD and MAX MEM GRAD parameters affect only gradient computation for locally connected and convolutional layers. Adjust these three parameters as needed to bring per-launch time under the Windows limit. The default values apply no limitation, which is good whenever possible, as breaking the task into multiple launches introduces significant overhead.

ANNEAL ITERS = Number

This is the number of simulated annealing iterations used to find good starting weights for refinement. The user can interrupt annealing by pressing the ESCape key, at which point refinement will commence with the best weights found so far. The default is 100.

ANNEAL RANGE = Number

This is the approximate range of random values tried at the start of simulated annealing. Larger value provide a wider search space but are also more likely to produce excessively large initial weights which can never be reduced to reasonable values. It's better to err on the side of too small than too large. The default is 0.1.

MAX ITERS = Number

This is the maximum number of conjugate gradient iterations used for weight refinement. The default is 1000. It may be good to set this to a smaller value if you are processing a collection of training operations in a single control file. However, in most cases it's best to make this a very large number and use the next parameter, *TOL*, to end training. Or you can manually interrupt training when the criterion graph looks like it has stabilized.

TOL = Number

This is the preferred method for determining convergence of the weight refinement algorithm. Roughly speaking, this specifies the degree of iteration-to-iteration criterion improvement for deciding that convergence is obtained. The default is 0.00005. Smaller values will force more extended training. Training ends when either *MAX ITERS* or *TOL* is hit.

WPEN = Number

This is the weight penalty, which penalizes large weights. A positive value will, by definition, degrade the performance criterion of the trained model. However, because large weights are often associated with overfitting, one may obtain better out-of-sample performance. The default is zero. A little weight penalty goes a long way, so if you experiment, start out very small, such as 0.001 or so.

Operations

As of now, there are three operations that can be performed with CONVNET within a control file. These are:

TRAIN

A model is trained using the current dataset. This operation can be roughly divided into four phases. In the first phase, simulated annealing is used to find good starting weights for subsequent refinement. Pressing the ESCape key interrupts annealing, and refinement will proceed with the best weights found so far.

The second phase is weight refinement using conjugate gradient optimization. This, too, can be interrupted with ESCape. However, in some cases the computer may take considerable time to respond, as certain sub-phases are not interruptible. Be patient.

The third phase is short, a final pass through the data with the best weights found. This phase can be interrupted with ESCape. However, doing so will cause all results to be lost. Be warned.

The fourth phase is computation of the confusion matrix. Unfortunately, the current version of CONVNET does not allow interruption of this operation. Patience is a virtue.

TEST

This assumes that a dataset as well as a trained model are present. Performance criteria, mainly the confusion matrix, are computed.

CLEAR

All data is erased, but a trained model, if present, is not disturbed. The usual purpose of this command is to allow reading of a test set after a model has been trained. The usual sequence is:
Read training data
Train
Clear
Read test data
Test

Display Options

Several options for displaying useful information as screen images are available. They are described in this section.

Display Training Images

Images from the training set are displayed. This option is enabled only if the images have one or three bands. The user enters the following information on a menu:

First to display

This is the ordinal number (1 is the first) of the first training set case to display. Images start in the upper-left corner of the screen and advance left to right first. If the total number to display exceeds the number in the training set, cases will wrap around to the first case in the training set.

Rows

This many rows of images will be displayed.

Columns

This many columns of images will be displayed. The total number of training cases displayed is *Rows* times *Columns*.

Display Filter Images

If the input image has either one or three bands, and a trained model exists, and the first hidden layer of this model is convolutional, this option displays filter weights as images. The displayed images have the same dimensions and orientation as the filter.

If the input image has one band, the display is black and white, with strongly negative weights being black and strongly positive weights being white. Intermediate weights are shades of gray.

If the input image has three bands, the display use a three-color display, with red, green, and blue matching the corresponding colors in the input image. For example, if the weights for all three bands are strongly negative, the corresponding image pixel will be black. If all three are strongly positive, the pixel will be white. A red pixel means that the weight for the red channel of the input image is strongly positive, and the weights for the other two channels are strongly negative. Et cetera. The user specifies the following parameters:

First slice to display

This is the ordinal number of the first slice to display. Images start in the upper-left corner of the screen and advance left to right first. If the total number to display exceeds the number of slices, they will wrap around to the first slice.

Rows for slices

This many rows of slice images will be displayed.

Columns for slices

This many columns of slice images will be displayed. The total number of slices displayed is *Rows* times *Columns*.

Scale slices individually

By default, the scale for mapping weights to tone is determined by examining all *Rows* times *Columns* displayed weights. If this box is checked, scaling is applied to each image separately, which may over-emphasize low-utility filters.

Display Activation Images

If a trained model and dataset are present, we can display the activations of the first hidden layer (any layer type) as images. The images are black and white, with black representing the lowest activation possible, and white the highest.

The user specifies the following parameters:

First slice to display

This is the ordinal number of the first slice to display. Images start in the upper-left corner of the screen and advance left to right first. If the total number to display exceeds the number of slices, they will wrap around to the first slice.

Rows for slices

This many rows of slice images will be displayed.

Columns for slices

This many columns of slice images will be displayed. The total number of slices displayed is *Rows* times *Columns*.

Case number

This is the ordinal number of the training case whose activations are displayed. It must not exceed the number of training cases.

Example of Displays

This section provides an example demonstrating the several display options that are available.

Figure 5.2 on the next page shows an example of the numeral zero taken from the MNIST dataset. A model consisting of a single convolutional layer having eight slices is created to train using the MNIST dataset. Figure 5.3 shows what the weights for each of these eight slices look like early in the training process. Note the great randomness. Figure 5.4 shows the same display after training has progressed to convergence. Note how clear response patterns have emerged. Finally, Figure 5.5 shows the activation pattern of the eight slices when presented with the MNIST zero of Figure 5.2.

It's worth pursuing this a little further. Look at the weight pattern in the second slice (top row, second from left) of Figure 5.4. It's very bright (high positive weights) near the center, and fairly or greatly dark (zero or negative weights) elsewhere. As one would expect, the activation pattern for the same slice in Figure 5.5 largely replicates the input image, though with some blurring.

Compare this with the last (bottom-right) slice. This weight set is just the opposite, being very dark (negative weights) in the center. We see in the corresponding activation display that the pattern is the negative of the input image. Lovely.

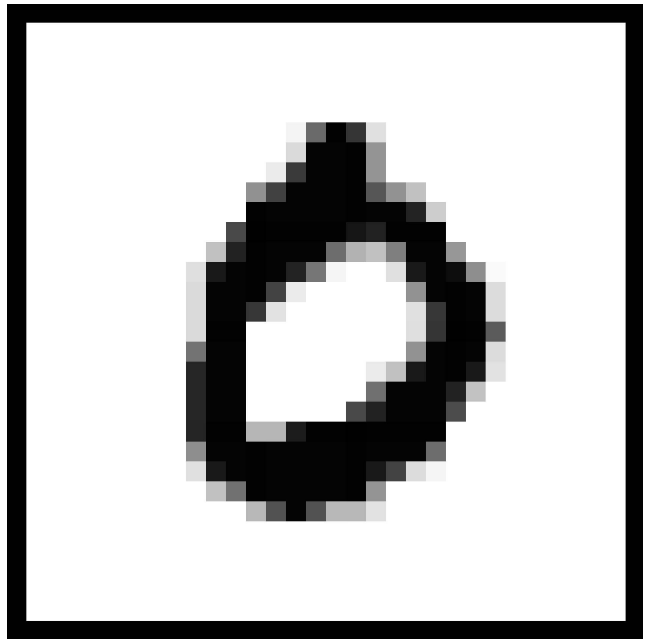

Figure 5.2: MNIST zero

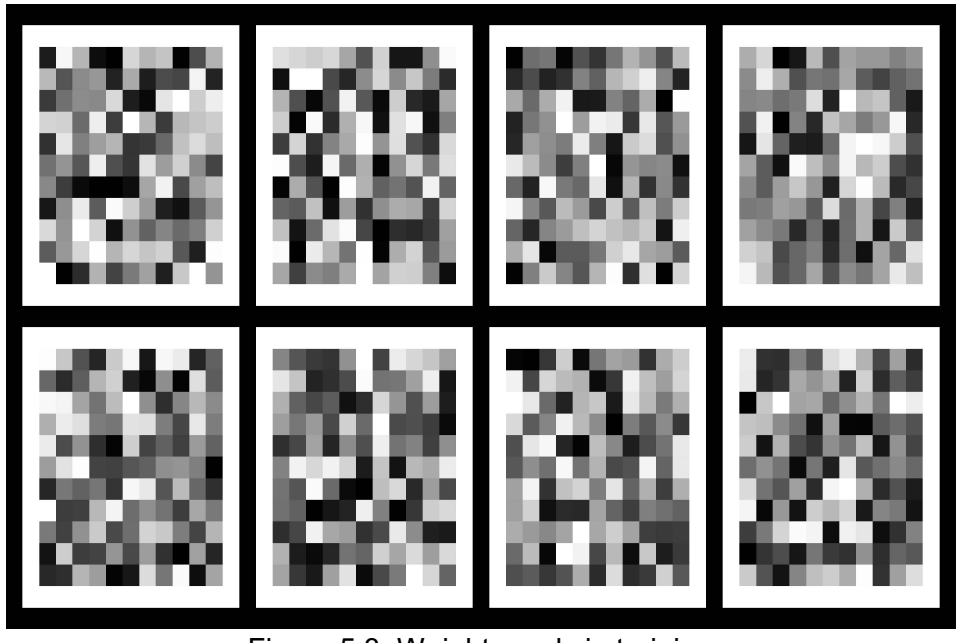

Figure 5.3: Weights early in training

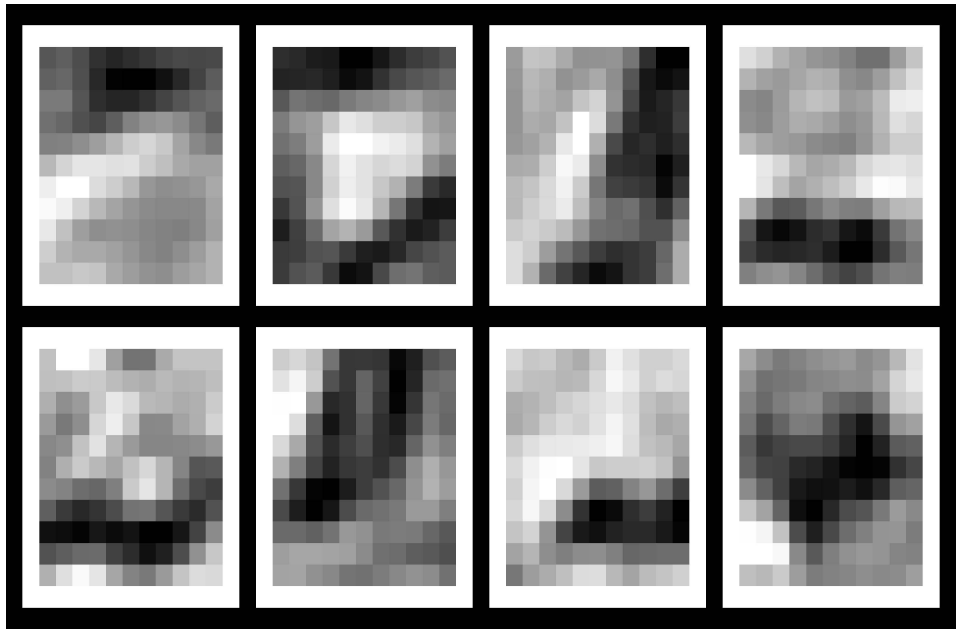

Figure 5.4: MNIST weights trained to convergence

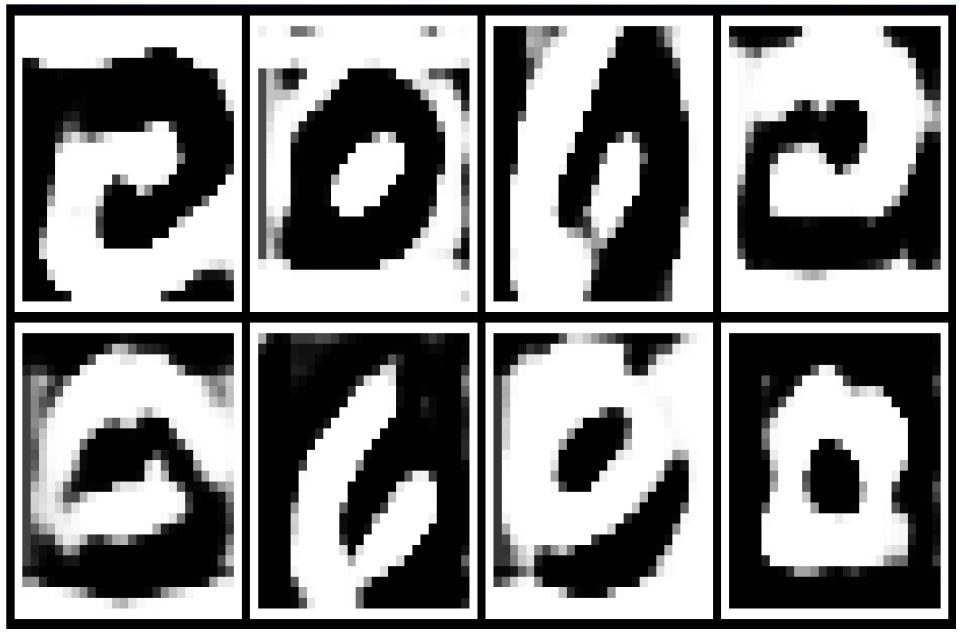

Figure 5.5: MNIST zero activations

The CONVNET.LOG file

The CONVNET program writes a log file that contains information about all operations. In order to understand this file, the following control file was created. It employs every available layer type.

```
MAKE IMAGE 12 12 1 6 1024
CONVOLUTIONAL LAYER  6 1 1 1 1 1 1
POOLED MAX LAYER 3 3 2 2
LOCAL LAYER  3 1 1 1 1 1 1
POOLED AVERAGE LAYER 3 3 2 2
FULLY CONNECTED LAYER 4
WPEN = 0.001
TRAIN
```

The log file echoes these lines, which we will skip here. The first important section in the log file is its description of the model's architecture.

```
Input has 12 rows, 12 columns, and 1 bands
Model architecture...
Model has 6 layers, including fully connected output
  Layer 1 is convolutional, with 6 slices, each 12 high and 12 wide
    Horz half-width=1, padding=1, stride=1
    Vert half-width=1, padding=1, stride=1
    864 neurons and 10 prior weights per slice gives 60 weights
  Layer 2 is 3 by 3 pooling max, with stride 2 by 2, 5 high, 5 wide, and 6 deep
  Layer 3 is locally connected, with 3 slices, each 5 high and 5 wide
    Horz half-width=1, padding=1, stride=1
    Vert half-width=1, padding=1, stride=1
    75 neurons and 55 prior weights per neuron gives 4125 weights
  Layer 4 is 3 by 3 pooling average, with stride 2 by 2, 2 high, 2 wide, and 3 deep
  Layer 5 is fully connected, with 4 slices, each 1 high and 1 wide
    4 neurons and 13 prior weights per neuron gives 52 weights
  Layer 6 (output) is fully connected, with 6 slices (classes)
    6 neurons and 5 prior weights per neuron gives 30 weights
  4267 Total weights for the entire model
```

Because the first layer (convolutional) has the padding equal to the half-width, and no striding, we see that it has the same visual field dimensions as the input layer. If necessary, review Equation (2.8) on Page 25. The layer has 12*12*6=864 neurons. The filter size is ((2*1+1)^2)*1+1=10. (The *1 is the depth of the prior layer, and the +1 is the bias term.) All neurons in the visual field share the same weight set, so the total number of weights for the layer is the filter size (10) times the number of slices (6).

Equation (2.8) gives the size of the second layer: (12-3+0)/2+1=5.

Layer 3 has the padding equal to the half-width, and no striding, so its visual field dimensions are the same as the prior layer. The filter size is ((2*1+1)^2)*6+1=55. There is a different weight set for each of the 5*5*3=75 neurons in this layer, giving a total of 4125 weights for this layer.

Equation (2.8) gives the size of the fourth layer: (5-3+0)/2+1=2.

Layer 5 is fed by 2*2*3 neurons in the prior layer. Including the bias term give 13 weights per neuron. This layer has 4 neurons, so it has a total of 52 weights. Recall that our convention is that fully connected layers have a 1*1 visual field, with a depth equal to the number of neurons.

Layer 6, the output layer is by definition fully connected. It's fed by 1*1*4 neurons in the prior layer. Including the bias gives 5 weights per neuron. It has a depth of 6, the number of classes, so it has 30 weights.

Adding these gives a total of 4267 weights in the model.

Simulated annealing completes, but I interrupted refinement. The following lines appear:

```
Simulated annealing for starting weights is complete with mean negative log likelihood =
0.29804

WARNING... User pressed ESCape during optimization
        Results are incomplete and may be seriously incorrect

Optimization is complete with negative log likelihood = 0.09214
```

The last item printed is a confusion matrix. The row (in groups of three) is the true class, and the column is the predicted class. In each set of three rows for a true class, the first row is the count, the second row is the percent for that row (true class) and the third row is the percent of the entire dataset.

	1	2	3	4	5	6
1	168	0	2	0	2	0
	97.67	0.00	1.16	0.00	1.16	0.00
	16.41	0.00	0.20	0.00	0.20	0.00
2	1	127	18	0	1	31
	0.56	71.35	10.11	0.00	0.56	17.42
	0.10	12.40	1.76	0.00	0.10	3.03
3	1	12	120	2	4	11
	0.67	8.00	80.00	1.33	2.67	7.33
	0.10	1.17	11.72	0.20	0.39	1.07
4	8	0	1	124	48	1
	4.40	0.00	0.55	68.13	26.37	0.55
	0.78	0.00	0.10	12.11	4.69	0.10
5	6	1	0	11	178	0
	3.06	0.51	0.00	5.61	90.82	0.00
	0.59	0.10	0.00	1.07	17.38	0.00
6	0	20	1	0	1	124
	0.00	13.70	0.68	0.00	0.68	84.93
	0.00	1.95	0.10	0.00	0.10	12.11

Total misclassification = 17.8711 percent

Printed Weights

The user has the option of printing weights for the entire model. Be warned that the total number of weights can be enormous, in which case the resulting file will also be enormous, and it may even require several minutes of run time to do the file writing. Here is a partial listing of the weights for the example cited in the prior section. Please reconcile this listing with the architecture of this model.

```
Layer 1 of 6 (Convolutional)  Slice 1 of 6
          3.642629  Input band 1 Neuron 1
         -0.676231  Input band 1 Neuron 2
         -0.085785  Input band 1 Neuron 3
          2.766258  Input band 1 Neuron 4
         -2.646048  Input band 1 Neuron 5
         -0.865142  Input band 1 Neuron 6
          1.900750  Input band 1 Neuron 7
         -2.298438  Input band 1 Neuron 8
          0.924283  Input band 1 Neuron 9
        ----------------------------------
         -3.971506  BIAS

... (Slices 2-5)

Layer 1 of 6 (Convolutional)  Slice 6 of 6
          3.011171  Input band 1 Neuron 1
          0.687377  Input band 1 Neuron 2
          1.019491  Input band 1 Neuron 3
         -0.832090  Input band 1 Neuron 4
          1.724954  Input band 1 Neuron 5
         -1.247742  Input band 1 Neuron 6
          0.444635  Input band 1 Neuron 7
          1.737460  Input band 1 Neuron 8
         -0.542140  Input band 1 Neuron 9
        ----------------------------------
         -2.507262  BIAS

Layer 2 of 6 (Mean pool) 5 rows by 5 cols by 6 slices
```

```
Layer 3 of 6 (Local)Slice 1 of 3 Row 1 of 5 Col 1 of 5
          0.016978  Prior layer slice 1 Neuron 1
         -0.027422  Prior layer slice 1 Neuron 2
         -0.052678  Prior layer slice 1 Neuron 3
          0.036557  Prior layer slice 1 Neuron 4
         -0.755227  Prior layer slice 1 Neuron 5
          0.211502  Prior layer slice 1 Neuron 6
          0.036439  Prior layer slice 1 Neuron 7
         -0.398360  Prior layer slice 1 Neuron 8
          0.737985  Prior layer slice 1 Neuron 9
        -----------------------------------

... Other rows and columns, then slice 2 and part of 3

Layer 3 of 6 (Local)Slice 3 of 3 Row 5 of 5 Col 5 of 5
         -1.035432  Prior layer slice 1 Neuron 1
         -0.357207  Prior layer slice 1 Neuron 2
         -0.021757  Prior layer slice 1 Neuron 3
         -0.033135  Prior layer slice 1 Neuron 4
         -0.107814  Prior layer slice 1 Neuron 5
         -0.000594  Prior layer slice 1 Neuron 6
         -0.051112  Prior layer slice 1 Neuron 7
          0.023901  Prior layer slice 1 Neuron 8
         -0.020555  Prior layer slice 1 Neuron 9
        -----------------------------------

... Slices 2 through 5
          0.679523  Prior layer slice 6 Neuron 1
         -1.053021  Prior layer slice 6 Neuron 2
          0.001994  Prior layer slice 6 Neuron 3
         -0.104741  Prior layer slice 6 Neuron 4
         -0.664431  Prior layer slice 6 Neuron 5
          0.034758  Prior layer slice 6 Neuron 6
          0.016724  Prior layer slice 6 Neuron 7
          0.014839  Prior layer slice 6 Neuron 8
          0.050983  Prior layer slice 6 Neuron 9
        -----------------------------------
         -1.963063  BIAS

Layer 4 of 6 (Avg pool) 2 rows by 2 cols by 3 slices
```

```
Layer 5 of 6 (Full)  Slice (this neuron) 1 of 4
          1.592443  Prior layer slice 1 Neuron 1
          1.161122  Prior layer slice 1 Neuron 2
         -0.162907  Prior layer slice 1 Neuron 3
          0.648188  Prior layer slice 1 Neuron 4
         -1.275991  Prior layer slice 2 Neuron 1
         -3.782788  Prior layer slice 2 Neuron 2
         -2.344005  Prior layer slice 2 Neuron 3
         -2.019643  Prior layer slice 2 Neuron 4
         -0.240221  Prior layer slice 3 Neuron 1
         -0.118739  Prior layer slice 3 Neuron 2
          0.739422  Prior layer slice 3 Neuron 3
          1.031370  Prior layer slice 3 Neuron 4
         -0.878146  BIAS

...
Layer 5 of 6 (Full)  Slice (this neuron) 4 of 4
          0.560776  Prior layer slice 1 Neuron 1
         -0.467746  Prior layer slice 1 Neuron 2
         -1.281872  Prior layer slice 1 Neuron 3
         -0.444215  Prior layer slice 1 Neuron 4
          0.948946  Prior layer slice 2 Neuron 1
          1.805807  Prior layer slice 2 Neuron 2
          1.796881  Prior layer slice 2 Neuron 3
          1.776497  Prior layer slice 2 Neuron 4
          4.415077  Prior layer slice 3 Neuron 1
          2.461983  Prior layer slice 3 Neuron 2
          2.944033  Prior layer slice 3 Neuron 3
          3.762620  Prior layer slice 3 Neuron 4
         -1.695120  BIAS

Layer 6 of 6 (Full)  Slice (this neuron) 1 of 6
          2.693996  Prior layer slice 1 Neuron 1
         -0.313751  Prior layer slice 2 Neuron 1
         -3.208661  Prior layer slice 3 Neuron 1
         -1.088728  Prior layer slice 4 Neuron 1
          0.714087  BIAS

...
Layer 6 of 6 (Full)  Slice (this neuron) 6 of 6
         -1.245246  Prior layer slice 1 Neuron 1
         -4.326880  Prior layer slice 2 Neuron 1
          1.525335  Prior layer slice 3 Neuron 1
          1.519400  Prior layer slice 4 Neuron 1
         -1.512020  BIAS
```

The CUDA.LOG File

CONVNET also writes a file called CUDA.LOG. It is divided into four sections. The first section names the CUDA device present and lists its capabilities. The second section lists the architectural and training parameters given by the user. The third section shows the device memory allocations, along with some supplementary information about allocation of convolutional gradient scratch memory. This may be of interest if device memory is limited and the user needs to tweak parameters to make optimal use of memory.

The last section is the most useful. It shows the total and per-launch device time, broken down by layer and by activity in each layer (forward-pass activation, backpropagation of delta, and computation of gradient). It also lists several other CUDA-related activities.

What makes this table important is the per-launch times. Windows imposes a limitation on this time. Currently, the default limit is two seconds. It can be changed with a registry hack, but you won't hear about it from me. The key thing is that this per-launch time lets the user tweak parameters. If gradient computation is the dominant per-launch issue, then the *Max CONV work per launch* parameter can be reduced. If activations are also a problem, the *Max batch size* parameter can be reduced.