Convolutional Neural Networks
and
Multi-Column Deep Neural Networks

I'm Arthur Dunbar by the way

Papers Covered

- This presentation takes ideas from 2 papers:
  - Gradient-Based Learning Applied to Document Recognition. LeCun, Bottou, Bengio and Haffner 1998
    - Mostly section II about Convolutional Neural Networks (CNN) and LeNet-5 (a specific CNN)
    - Stuff after section II is comparing it to other algorithms of the day, and explaining how to do those, and some stuff about the specifics of implementing it.
    - I skip this except a little about section III where I talk about their results
  - Multi-Column Deep Neural Networks for Image Classification. Ciresan, Meier, and Schmidhuber 2012
    - this paper that builds on the idea of CNNs
    - This is about MCDNNs which are basically an ensemble of C/DNNs
    - It still holds state of the art results on several data sets related to image classification
- Also of interest is: http://deeplearning.net/tutorial/lenet.html
  - This gives theano code for a more up to date version of LeNet (some parts are on other pages, but that is because they are shared between algorithms)
  - It is the source of some pictures I use
  - It also helped a lot in explaining just what was going on and connecting the dots in terminology changes between the first and second paper

Gradient-Based Learning Applied to Document Recognition

This paper is big and covers a lot of things, but we are interested in section II predominantly.

- Section II is “Convolutional Neural Networks for Isolated Character Recognition”

Section II – Convolutional Neural Networks for Isolated Character Recognition

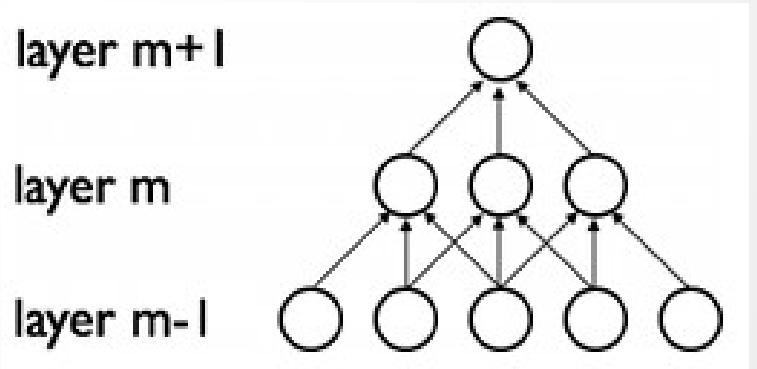
- Multi-layer neural networks trained with gradient descent can learn complex high-dimensional, non-linear mappings
  - This makes them seem like an obvious choice for image recognition
- While it can be done with traditional fully connect feed forward networks there are issues

Reasons Not to Use Traditional Feed Forward Networks on Images

1) Images are large with many many many pixels (variables).
   - A fully connected NN with 100 hidden layer nodes would be at least 10s of thousands of weights to calculate if not much more
     - This increases the capacity of the system, and that means that you need huge amounts of training data not to mention the memory required to store it
2) No built in translation or distortion invariance
   - Before being sent to the fixed-size input layer, images must be approximately size normalized and centered on the input field.
     - This is hard
   - Also many things are not quite standardized in real life. No two people draw the letter A the same, and this would cause the network to suffer
3) The way it would do the task if by some miracle it did do it correctly is a way that is a near miss of a much easier to train model, the **convolutional neural network**
4) Fully connected architectures ignore the topology of the input
   - Images have strong 2D local structure, and that is completely ignored
   - Local features are invaluable in image classification
     - Examples are edges and corners

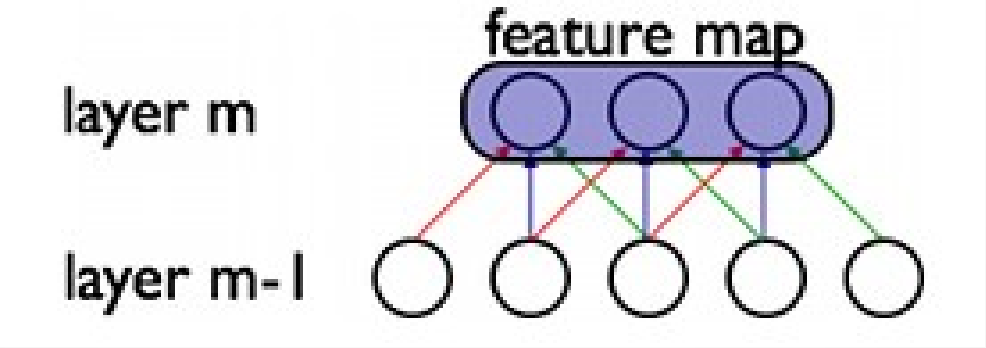
Convolutional Neural Network Basics

- **Convolutional Neural Network** (CNN)
  - Fun fact: Modeled after the image processing part of the brains of cats
  - Designed for learning images
  - Described as:
    - 'A specialized NN architecture that incorporates knowledge about the invariance of 2D shapes by using local connection patterns and by imposing constraints on weights.'
    - A deep neural network that works by convolving features (similar to how image filters are used in image processing)
  - Feed forward neural networks have all nodes in this layer connect to all nodes in the next layer. CNNs, however, only connect nearby nodes like so:
  - One reason CNNs are good is they avoid having to make hand crafted feature extractors
  - Note: This is shown as 1D, but images are 2D so instead of 3 it would be 3x3



CNN Basics continued

- CNNs
  - Each layer has some set of **feature maps**
    - Feature maps are each their own representation of the image but processed (like after convolution)
    - Within a feature map, each weight is shared with ones that point the same way (the red lines in the image below are the same weight as each other for instance)
    - These weights are learned at the same time



CNN Basics continued

- CNNs have three architectural ideas to ensure a degree of shift, scale and distortion invariance. Allowing the image to move and be different, sizes, etc to an extent.
  1. Local **receptive fields** – small portions of the image that act as local filters over the input space which are combined in subsequent layers to detect higher order features
     - For instance, a 5x5 section of pixels is used to calculate something about an output pixel (like in convolution). These are used for convolutional layers and sub-sampling layers
  2. Shared weights – Allows the pattern to be found anywhere. This is why all the red lines had the same weight last slide (it's also easier on the computer)
  3. Spacial / temporal sub-sampling – Sub-sampling allows for local info to be used for features.
     - One example used in the paper is taking every pixel in a 2x2 field, finding the most intense one and taking that to be the representative pixel. http://deeplearning.net/tutorial/lenet.html Calls this “Max Pooling” and so does the second paper, but this one calls it sub-sampling

CNN Layers

- There are several kinds of layers in a CNN. In the case of LeNet-5 there are the following layer types:
  - Convolutional
  - Sub-sampling / max pooling
  - Feature
  - Output
- There are multiple convolutional and sub-sampling layers, but there is only one feature and output layer each

The Convolution Layer

- For convolutional layer M
  - Each pixel is calculated by taking the pixels within the receptive field in layer M-1 and performing a dot product with a weight vector and adding a bias
    - 5x5 input field is like a 25 input perceptron with a bias
  - These overlap, so a pixel and one to its left with a 3x3 receptive field will share 6 pixels (but each pixel will share a weight with a different pixel from the other group)
  - This is done multiple times making multiple weighted convolved copies of the M-1 layer

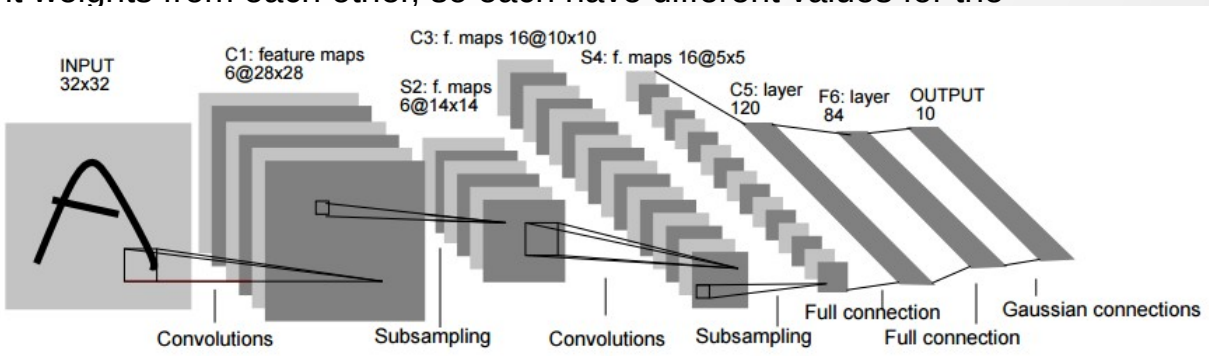
Sub Sampling / Max Pooling Layers

- Sub Sampling / Max Pooling Layers
  - Unlike the convolution layers, these do not overlap
  - Why are these layers here?
    1) Reduces computation for upper layers
    2) Provides translation invariance ('smart' robustness)
       - Normally there are 8 directions that a pixel can be translated each changing the output
       - With a 2x2 receptive field 3/8 of these give the exact same output as before. If it is 3x3 5/8 of them do when you cascade it with a convolution layer
         - http://deeplearning.net/tutorial/lenet.html Says this, but they don't explain it

A Specific CNN: LeNet-5

- Used to recognize characters (Like 'A' or '1')
- This algorithm performs well for small numbers of things to discriminate between, but it gets larger with more targets. It is historically important and relatively simple to explain however.
- It uses 32x32 pixel images and has 7 Layers:

1) **Convolutional layer** 1: uses 6 feature maps generated with a 5x5 receptive field (making it 6 28x28 maps)
   - With 5x5 as the receptive field, the new point is at the 3rd by 3rd pixel (the center) modified by the convolution. As said before these regions overlap.
   - This means that 2 pixels along the edge are removed because there aren't enough neighboring pixels to calculate them. (It doesn't guess or assume a value for them)
   - Each of the 6 feature maps have different weights from each other, so each have different values for the same location.



LeNet-5 continued

2) **Sub-sampling layer** 2:

- Uses a 2x2 receptive fields on each of the 6 28x28 maps making them 14x14 (reminder: does not overlap)

3) Convolution layer 3:

- Another 5x5 receptive field is used on each of the 14x14 maps above (to make 10x10)
- Several of these are combined making 16 10x10 images
- This is done by connecting the 5x5 regions across maps via this table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

4) Sub-sampling layer 4:

- 2x2 receptive field to make 5x5 images. There are still 16 of these

5) Convolution layer 5:

- The 16 layers are mapped to 120 features maps
- Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Because S4 is 5x5 this means each of the 120 is 1x1
- It's being named a convolutional layer is said in the paper to be a sort of misnomer, but it uses the same architecture as one, so it is called one. Also if the original image was bigger it would be one.

6) **Feature layer** 6:

- 84 nodes completely connected to the 120 nodes in CL5. The 84 is explained later.
- This is 10,164 parameters (121*84, each of the 84 has a bias)

7) **Output**

- Explained later

LeNet-5 continued

- All non output layers, like a normal feed forward neural network, pass the output of the perceptron through a squashing function.
  - $x_i = f(a_i)$ where i is the ith node and a is the weighted sum
- The squashing function, instead of a sigmoid, is a scaled hyperbolic tangent (not that unusual in NNs)
  - f(a) = A*tanh(S*a)
    - Where A and is a scale that is set to 1.7159
    - S is to change the gradient intensity
    - Why 1.7159? Because with that scale +/- 1 is the point of maximum curvature
      - This somehow avoids saturation by making things maximally non linear.
      - Saturation is bad because it makes learning slow

LeNet-5's output

- Output uses the **Euclidean Radial Basis Function** (RBF)
  - $y_i = \Sigma (x_j - w_{ij})^2$

    Where y_i is the output of node i
    x_j is the input of node j
    w_{ij} is not explicitly stated, but it is probably the parameter vector
    The summation is over all j
- This is the Euclidean distance between the input vector the **parameter vector**. The further from the parameter vector, the higher the RBF
- This is a penalty term measuring the fit between the input pattern and a model of the class associated with the RBF
- In probability terms, the RBF output is the unnormalized negative log-likelihood of a Gaussian distribution in the space of configurations of layer F6

LeNet-5's Loss Function

- Somehow things have to be penalized and rewarded to improve performance
- “Given an input pattern, the loss function should be designed to get the configuration of F6 as close as possible to the parameter vector of the RBF that corresponds to the pattern's desired class”
- The pattern vectors are chosen by hand and initially are fixed. They could have been random, but they decided to use a stylized image of the corresponding character class drawn on a 7 x 12 bitmap (why there are 84 feature layer nodes)
  - There are 96 examples, they are below
- This doesn't do too well on isolated digits because of easily confusable characters like 0 and O having similar output codes, but it does well for strings of digits if a linguistic post processor is used to pick the correct one from context.

Fig. 3. Initial parameters of the output RBFs for recognizing the full ASCII set.

LeNet-5's Loss Function

- The simplest loss function that could be used is the Maximum Likelihood Estimation (MLE) which is equivalent to Mean Squared Error (MSE).

$$E(W) = \frac{1}{P} \sum_{p=1}^{P} y_{Dp}(Z^p, W)$$

  - y_{Dp} is the output of the Dp-th RBF unit (the one that corresponds to the correct class of input pattern Z^p).

Loss Function continued

- That would have been good but it lacks 3 things

1)If we allow the parameters of the RBF to adapt, E(W) has a trivial but unacceptable solution

  - If all the RBF parameters are equal and the state of F6 is constant and equal to the parameter vector the network will ignore the input and all RBF outputs are 0
  - This doesn't happen if the RBF weights can't adapt

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}(y_{D^p}(Z^p, W) + \log(e^{-j} + \sum_i e^{-y_i(Z^p, W)}))$$

Loss Function continued

2)There is no competition between classes

- To make this work use instead the MAP (maximum a posteriori) function to maximize posterior probability of the correct class Dp (or minimizing the logarithm of the probability of the correct class)
  - This pushes down the penalty for the correct class and pulls up for incorrect
- Not only can it pick one but there is a 'rubbish' class label.
  - The posterior probability of this rubbish class is e^-j / (e^-j + Σ_i e^-yi(Zp,W))
- The negative of the second term (below in picture of equation) plays a competitive role, and it is smaller than the first term
- The constant j is positive and prevents the penalties of classes that are already large from being pushed further.
- All of this prevents collapsing by “keeping the RBF centers apart from each other”

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}(y_{Dp}(Z^p, W) + \log(e^{-j} + \sum_i e^{-y_i(Z^p, W)}))$$

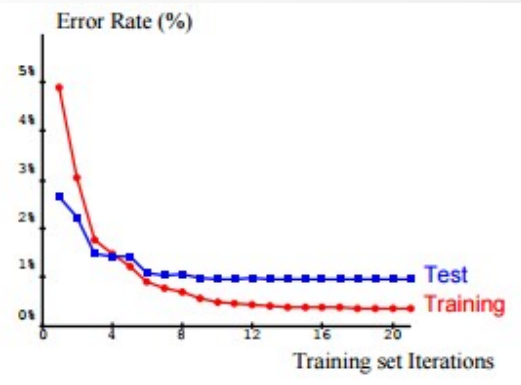
Loss Function continued

3)It doesn't explicitly say what 3 is. My best guess is that it refers to this though:

- The network has to use back propagation, but it has to be modified to account for weight sharing
- One way to do this is to compute the partial derivatives of the loss function with respect to each connection as if there was no weight sharing, then the partial derivatives of all the connections that share the same parameter are added to form the derivative with respect to that parameter

Paper 1 end (for this presentation) and LeNet-5 end

- This architecture can be trained efficiently, and it gives examples of how to do it in the appendix
- The rest of the paper is concerned with explaining things that are not LeNet-5 and some of them aren't CNNs
- As for the performance of LeNet-5:
  - According to wikipedia, it does well classifying digits, but if you give it to many characters it starts to have problems because the network has to get big even with all the things we just covered.
  - According to the paper it had about a slightly lower than 1% error rate on the test set
  - Left diagram is the misclassified examples right is the training vs testing error with iterations



Any questions on Paper 1 or CNNs in general?

Multi-Column Deep Neural Networks for Image Classification (MCDNN)

- This paper focuses on a specific kind of Deep Neural Network that uses 'columns' that work independently. Each is a Deep neural network itself
- This paper assumes you know things covered in the previous paper as it uses terms like convolutional and receptive field without telling you what that means
  - By this I don't mean it says read that paper first, it just sort of assumes you know what receptive field mean
- MCDNN is optimized for being used with GPUs
  - If you don't know Graphics Processors are good for certain parallel programing tasks because they tend to have large vector adders that can do many operations at once
  - In this, for example, they claim 2 orders of magnitude speed up using GPUs allowing them to train it in hours or days

MCDNN intro continued

- Training is fully online with weight updates occurring after each back propagation of errors
- They claim that properly trained, wide and deep DNNs can outperform all previous methods (paper from 2012)
- They also claim that unsupervised initialization and pretraining is not necessary (though sometimes useful if there are few examples of each class)
- They also show that adding the columns to the DNN to make it a MCDNN reduces error rates by 30 to 40%

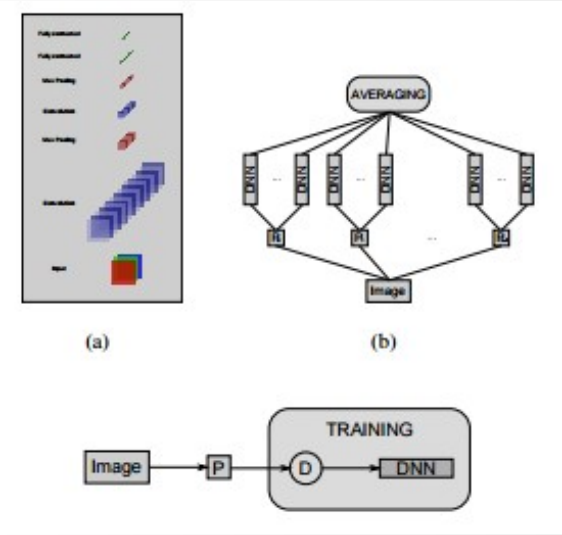
MCDNN Architecture

- Initial weights are random, and they are trained on a training set and tested on a testing set (duh). They do this in a novel way though

1) Unlike many small NNs which are shallow or LeNet-7 (an update to LeNet-5 from earlier), theirs has hundreds of maps per layer with 6 to 10 layers of non-linear neurons stacked on top of each other
    - They wind up with a number of connections that is comparable to the visual system of the Macaque monkey
2) It has been shown that DNNs are hard to train by gradient descent, the mathematically preferred way, but computers are 60,000 times faster now than in the 90s, and GPUs allow for even more speed up. Given enough data, their network does not need additional heuristics like unsupervised learning

MCDNN Architecture continued

3)The DNN showed here (a)-> has 2D layers of winner take all neurons with overlapping receptive fields.



- “Given some input pattern, a simple max pooling technique determines winning neurons by partitioning layers into quadratic regions of local inhibition selecting the most active neuron in each region”
- “The winners represent a down-sampled layer with lower resolution feeding the next layer in the hierarchy”
- Basically the sub-sampling layer from LeNet-5

MCDNN Architecture continued

4)Down sampling (max pooling, sub sampling etc) eventually leads 1D layers. From then on only trivial winner take all 1D regions are possible.

- Aka, it is now a standard multi-layer perceptron
- Receptive fields and winner take all fields of a DNN are often near minimal (2x2, 3x3) to achieve near maximal depth of layers with non trivial regions

MCDNN Architecture continued

5)Only winning neurons are trained

- Meaning losing neurons don't forget what they've learned so far
- This corresponds to a biological reduction of energy consumption

MCDNN Architecture continued

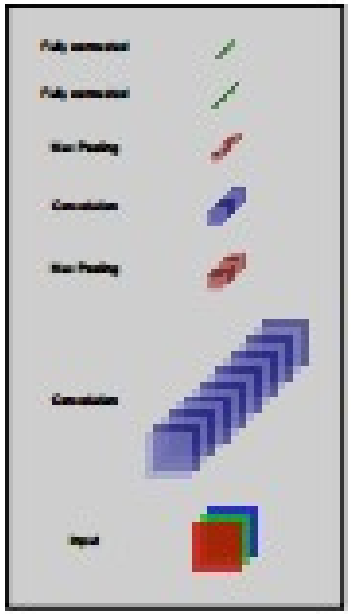
6)Inspired by 'microcolumns' of neurons in the cerebral cortex, they combine several DNNs to form a MCDNN

- Given some input pattern, the predictions of all columns are averaged
- Various columns can be trained on the same input or different pre-processings of the input
  - Using different pre-processings reduces errors and reduces work

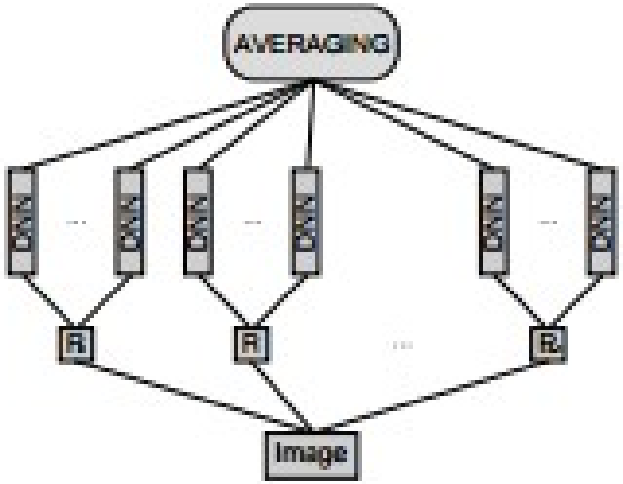
$$y^i_{MCDNN} = \frac{1}{N} \sum_{j}^{\#columns} y^i_{DNN_j}$$

MCDNN Architecture

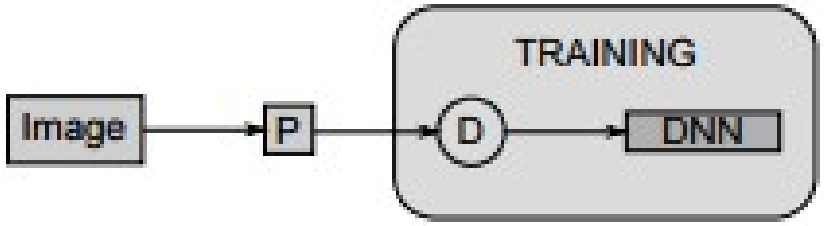
- (a) is a DNN
- (b) MCDNN
  - Where each DNN is (a
  - And the Ps are pre-processings shared by a few DNNs
- (c) training a DNN
  The image is preprocessed (p) and distorted (d)



(a)



(b)



Experiments – describing DNNs

- The paper describe DNNs like this for the experiments
  - 2x48x48-100C5-MP2-100C5-MP2-100C4-MP2-300N-100N-6N
    - 2 input images of size 48x48
    - A convolutional layer with 100 maps and 5x5 receptive field
    - A max pooling layer with 2x2 receptive field
    - Another 100 map convolutional layer but with 4x4 receptive field
    - Another max pooling layer with 2x2 receptive field
    - Fully connected 300 neuron hidden layer
    - Fully connected 100 neuron hidden layer
    - Fully connected output layer with 6 neurons

DNN training details

- All DNN are trained using on-line gradient descent with an annealed learning rate
  - it later says in a specific example” i.e. initialized with .0001 multiplied by a factor of .993/epoch until it reaches .00003”
  - Annealing is slowing down a learning rate though
- During training, images are continually translated, scaled and rotated, and in the case of characters elastically distorted
- Only the original image is used for validation though
- Training ends when the validation error is zero or when the learning rate reaches a predetermined minimum
- Initial weights are randomly chosen from [-.05, .05]

The MCDNN they used on MNIST

- MNIST is a data set of digits
- Wikipedia has a page on the MNIST data set and lists the best algorithm out there on it as this one still with 0.23% misses
- MNIST digits are normalized such that the width or height of the bounding box is 20 pixels
  - Aspect ratios vary strongly, so they create six additional data sets by normalizing digit width to 10, 12, 14, 16, 18, 20 pixels
  - This is an attempt to make it like they see it from different angles etc.
- They train 5 DNNs per normalization and the original and put these into the MCDNN for 35 columns

MCDNN for MNIST continued

- Each DNN is 1x29x29-20C4-MP2-40C5-MP3-150N-10N
  - And each is trained for 800 epochs with an annealed learning rate
  - This takes 14 hours and after 500 epochs little additional improvement is observed
- During training the digits are randomly distorted before each epoch

MCDNN for MNIST

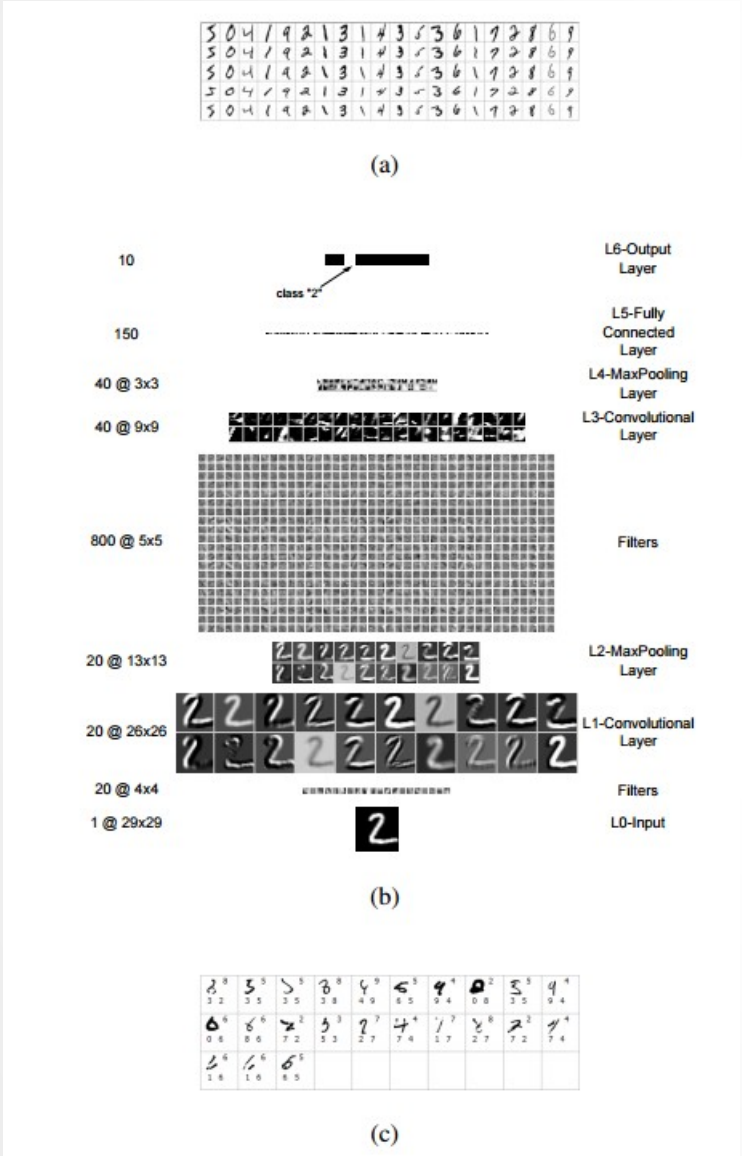


Figure 2. (a) Handwritten digits from the training set (top row) and their distorted versions after each epoch (second to fifth row). (b) DNN architecture for MNIST. Output layer not drawn to scale; weights of fully connected layers not displayed. (c) The 23 errors of the MCDNN, with correct label (up right) and first and second best predictions (down left and right).

MNIST results

Table 1. Test error rate [%] of the 35 NNs trained on MNIST. Wxx - width of the character is normalized to xx pixels

Trial	W10	W12	W14	W16	W18	W20	ORIGINAL
1	0.49	0.39	0.40	0.40	0.39	0.36	0.52
2	0.48	0.45	0.45	0.39	0.50	0.41	0.44
3	0.59	0.51	0.41	0.41	0.38	0.43	0.40
4	0.55	0.44	0.42	0.43	0.39	0.50	0.53
5	0.51	0.39	0.48	0.40	0.36	0.29	0.46
avg.	0.52±0.05	0.44±0.05	0.43±0.03	0.40±0.02	0.40±0.06	0.39±0.08	0.47±0.05
	35-net average error: 0.44±0.06						
5 columns MCDNN	0.37	0.26	0.32	0.33	0.31	0.26	0.46
	35-net MCDNN: **0.23%**						

Table 2. Results on MNIST dataset.

Method	Paper	Error rate[%]
CNN	[32]	0.40
CNN	[26]	0.39
MLP	[5]	0.35
CNN committee	[6]	0.27
MCDNN	this	**0.23**

- Table 1 (left) shows the results of each DNN in the MCDNN at the top and the total MCDNN below. In all but one category the MCDNN beats every constituent part. These vote to get .23% error
- Table 2 (right) shows it against other algorithms from other papers, and it claims a 35% improvement over state of the art
- They mess up 23 total, of these 20 had correct second guesses, and most of them were highly irregular or wrongly labeled

Other Datasets NIST SD 19

- NIST SD 19 Latin characters, they beat everyone they could find. Wikipedia doesn't have a page on this one, and I couldn't find a central results page

Table 4. Average error rates of MCDNN for all experiments, plus results from the literature. * case insensitive

Data (task)	MCDNN error [%]	Published results Error[%] and paper	
all (62)	**11.63**		
digits (10)	**0.77**	3.71 [12]	1.88 [23]
letters (52)	**21.01**	30.91[16]	
letters* (26)	**7.37**	13.00 [4]	13.66[16]
merged (37)	**7.99**		
uppercase (26)	**1.83**	10.00 [4]	6.44 [9]
lowercase (26)	**7.47**	16.00 [4]	13.27 [16]

Other Datasets and Conclusion

- The rest of the paper other than conclusions is about other data sets and some specifics about how many layers have how many nodes with various receptive fields etc. It's interesting, but not presentation material.
  - Here's there results compared with the best method other than theirs they could find with a percentage improvement
  - Table 4 is on the previous page
  - The conclusions part is basically a we did well and MCDNNs are good statement

Dataset	Best result of others [%]	MCDNN [%]	Relative improv. [%]
MNIST	0.39	0.23	41
NIST SD 19	see Table 4	see Table 4	30-80
HWDB1.0 on.	7.61	5.61	26
HWDB1.0 off.	10.01	6.5	35
CIFAR10	18.50	11.21	39
traffic signs	1.69	0.54	72
NORB	5.00	2.70	46

The End
Questions?