Image Classification of Letters

- A Model Comparison Approach -

•••

General Assembly Final Project 2018

by Mundy Otto Reimer

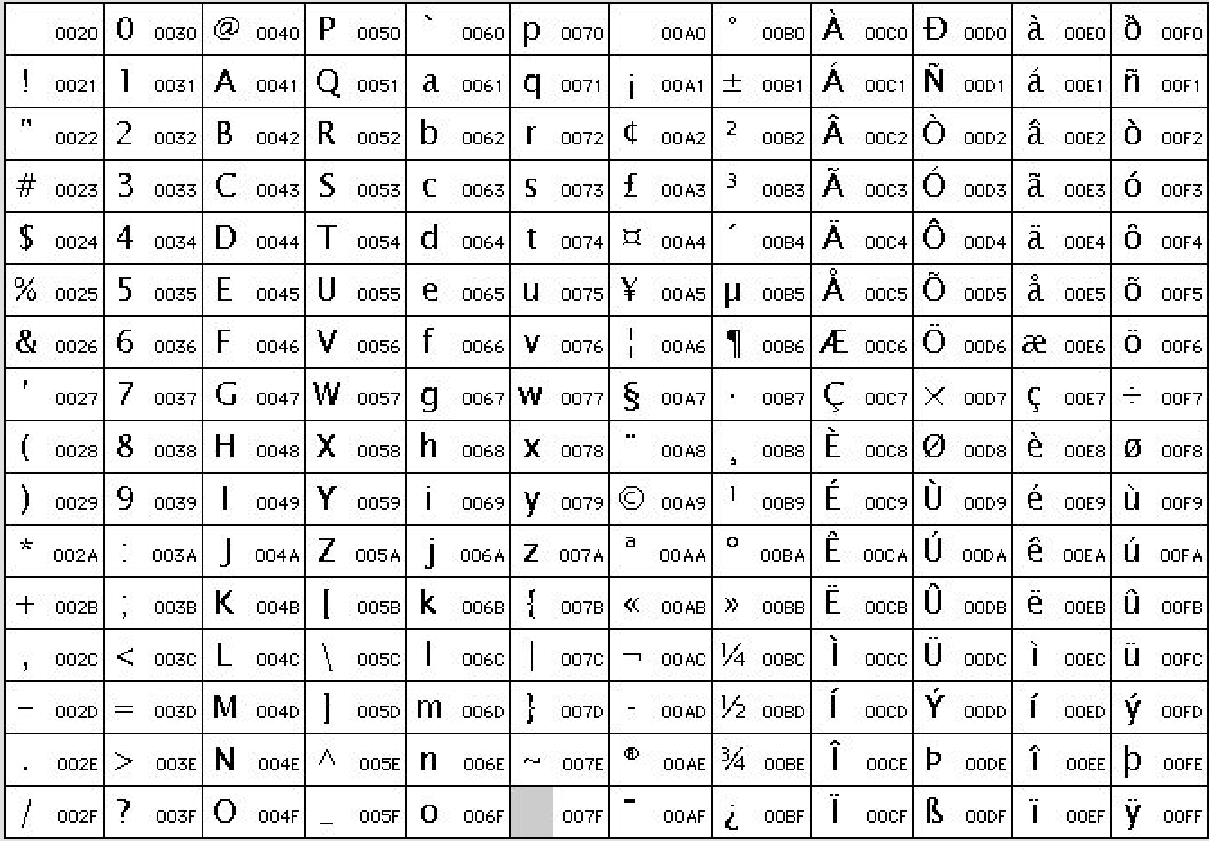
notMNIST Dataset

- 10 classes for letters A - J
- Different Glyphs of Fonts
- Size 28x28 pixels
- 19k hand-cleaned instances
- 500k uncleaned instances

Objective:

Classification Problem

.png Image -> Unicode Value



0020	0 0030	@ 0040	P 0050	` 0060	p 0070	00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
" 0022	2 0032	B 0042	R 0052	b 0062	r 0072	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
' 0027	7 0037	G 0047	W 0057	g 0067	w 0077	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
(0028	8 0038	H 0048	X 0058	h 0068	x 0078	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
- 002D	= 003D	M 004D	] 005D	m 006D	} 007D	- 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
/ 002F	? 003F	O 004F	_ 005F	o 006F	007F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Cleaning the Data

Loading

- Un-Pickling
- Each class loaded into separate dataset (memory issues)
- Merged at end in 1 big data set

Normalizing

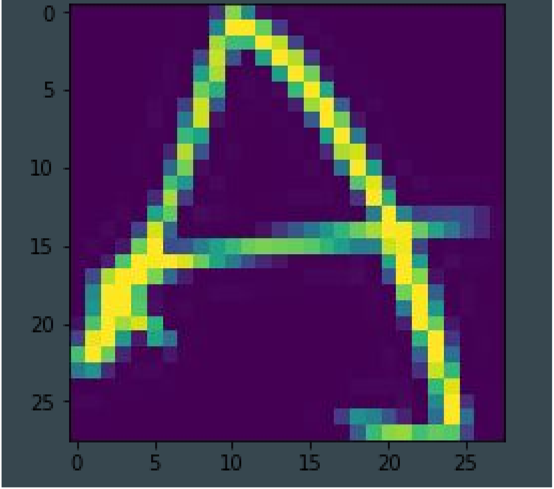
- Convert Dataset into 3D array

(image index, x, y)
Floating Point #s

- Normalized to ~zero mean & std ~0.5 (easier training)

Sampling Labeled Data

- Verify data & labels still ok



Partition & Process Data: Training > Validation > Test

Class Distribution

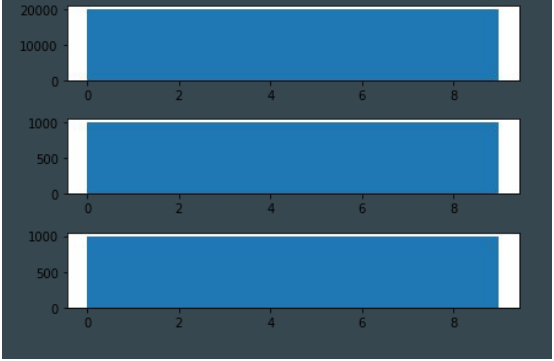
- Check data if balanced across letter classes (Accuracy Paradox)

52k per Large Letter

2k per Small Letter

Randomization

- Shuffle labels so training and test distributions match
- Plot each histogram



Measure Overlap

- After split of 90/5/5, check for Training, Validation, Test sample overlap

# Duplicates in training set ~12k

# Duplicates in train+val ~900

# Duplicates in train+test ~1k

- Expect to use in environment w/ no overlap?

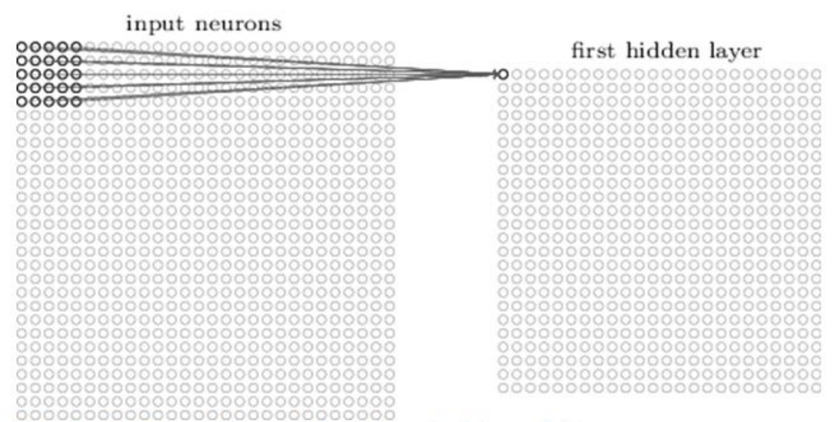
Model Comparisons

	Training / Val / Test Scores	
Logistic Regression	.76 / .80	● Cheap & Simple ● Scores high (90s) for classic MNIST dataset
LR w/ Gradient Descent	.78 / .75/ .82	● Used softmax at end (R -> [0,1]) for multiple classes
LR w/ Stochastic Gradient Descent	.77 / .78 / .85	● 4.76 sec (as opposed to 15+ sec) ● Little improvement in accuracy
1-Layer Feedforward Neural Network	.83 / .82/ .88	● Small accuracy improvement over faster & simpler LR w/ Stochastic GD ● Computationally Expensive AND opaque

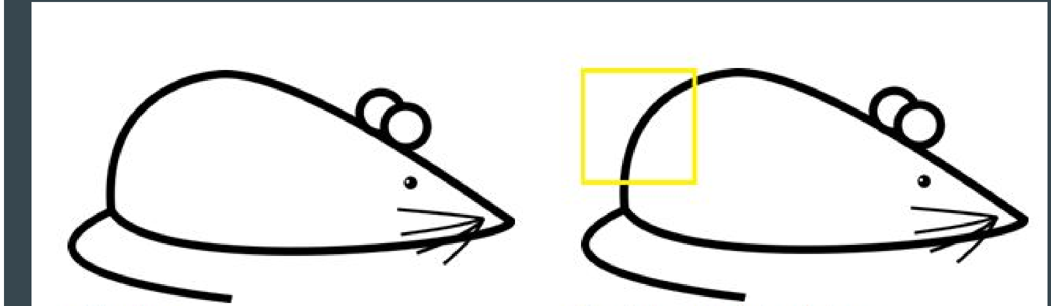
Model Comparisons

	Training / Val / Test Scores	
Neural Network w/ L2 Regularization	.86 / .85 / .91	● L2 Ridge (Used b/c large weights indicate possible overfitting?), so multiply them by small fraction (adds penalty on norm of the weights to loss)
Neural Dropout	.84 / .85 / .91	● Since improvement from overfitting, try another overfitting technique (didn't improve much) ● Also used on fully-connected networks
Multiple Layers & Learning Rate Decay	.86 / .85 / .91	● Multiple layers too computationally expensive (+hours) ● Instead implemented LRD to reduce training time back to ~10 sec
2+1-Layer Convolutional Neural Network	.69? / .86 / .93	● Computation time went back up (40 min) ● Stride 2 to reduce dimensionality (13 min)

Convolutional Neural Networks



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

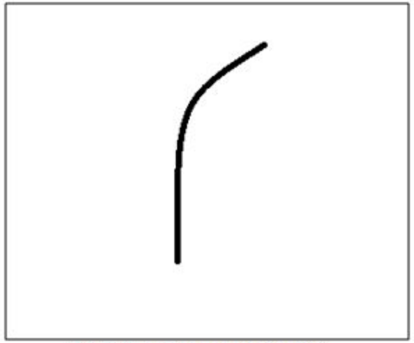


Original image

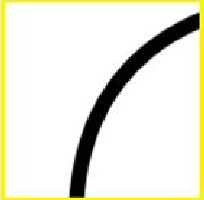
Visualization of the filter on the image

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

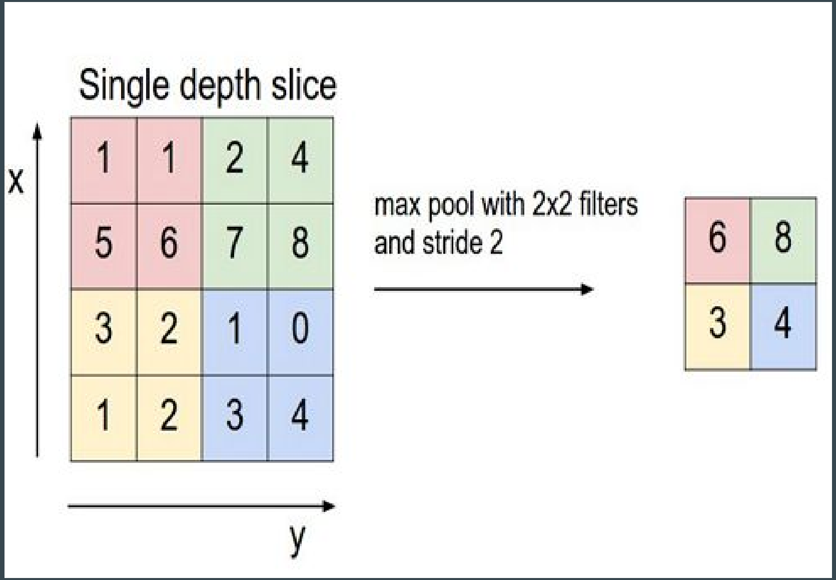
*

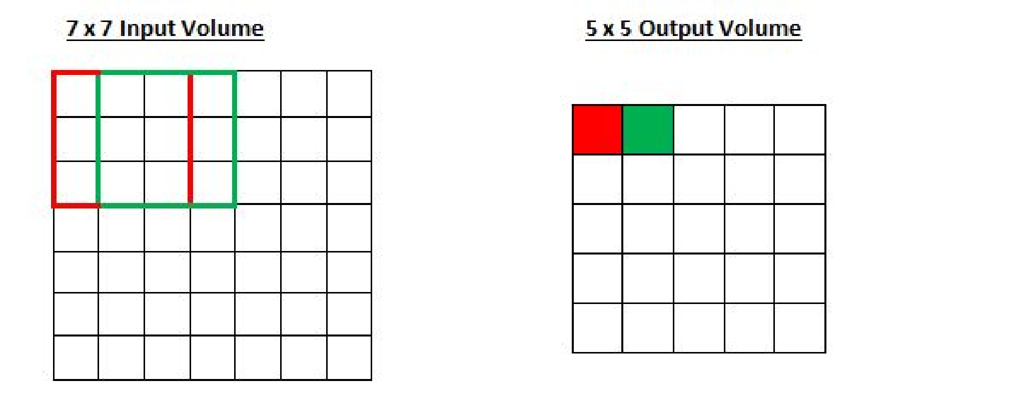
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

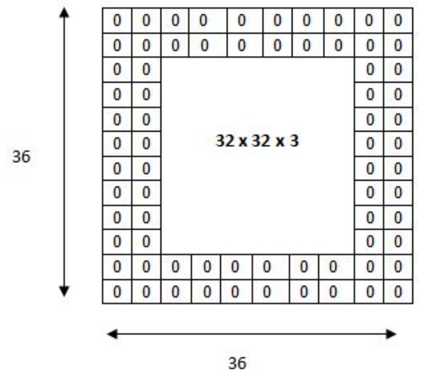
Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

CNN cont: Stride & Padding



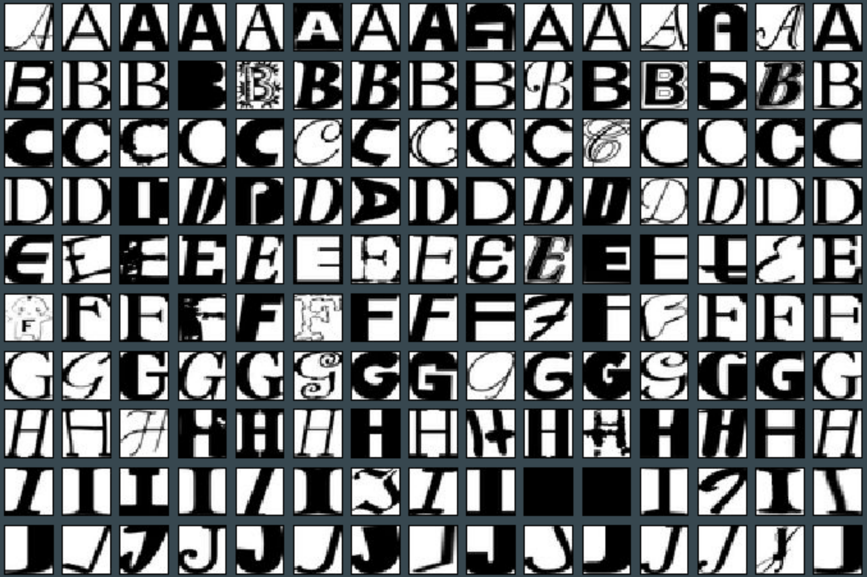




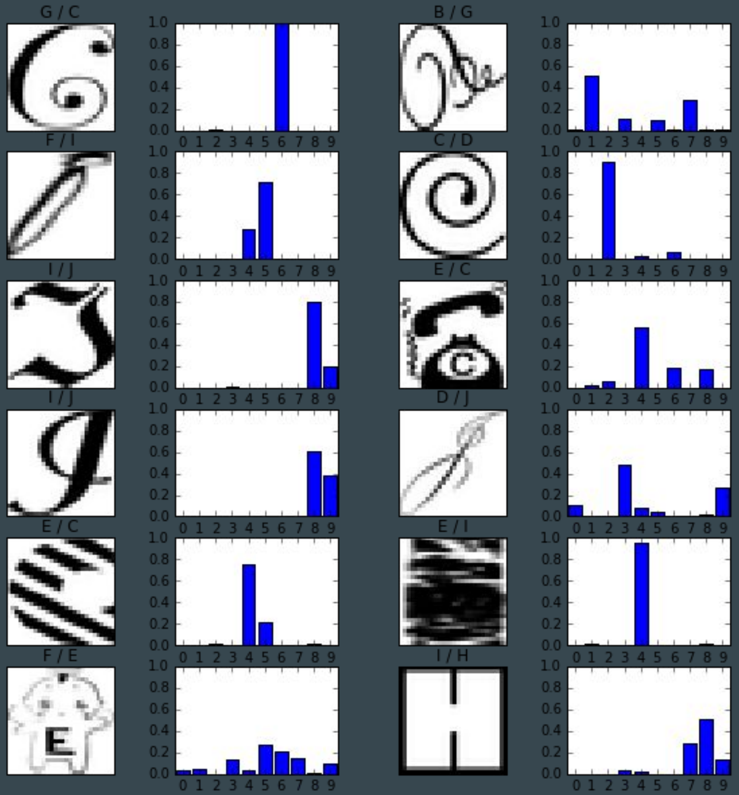
The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

Problems (from CNN output)

Categorized Correctly



Incorrect Categorizations & Confidence Level for each Letter



Sources

My Git Repo & Jupyter Notebook:
https://github.com/mundyreimer/notMNIST_project/blob/4f2e7b3918f585933995d6c3d227a66e6711fbb1/projects/final-project/04-notebook-rough-draft/notMNIST.ipynb

Dataset: by Yaroslav Bulatov

http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html

Tensorflow Tutorials:
http://nbviewer.jupyter.org/github/jdwittenauer/ipython-notebooks/tree/master/notebooks/tensorflow/

Theoretical Explanation of Handwritten Digit Recognition:
https://faisalorakzai.wordpress.com/2016/06/01/handwritten-digits-recognition-using-deep-learning/

Beginner's Guide to Understanding Convolutional Neural Networks:
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Thanks!