Convolutional Neural Networks for Classification of Data in High-Dimensional Datasets

Nishanth Bhaskara
Mentors: George Djorgovski, Ciro Donalek

Abstract:

Machine learning is an important technique for data analysis and pattern detection in complex or large data sets. Certain datasets are very high in dimensionality and so most existing machine learning techniques fail to scale in feasibility. Although methods such as feature selection or feature extraction exist, they fail to look at the data as a whole. Our research is mainly focused on using CNNs, a neural network which utilizes filters that convolve over the input vector, to try and provide a generic solution to the classification problem of any high dimensional data set. As a proof of concept, the research is geared towards video classification, namely fMRI clips of patients either testing positive or negative to autism. The fMRI clips are different from normal images as they add an extra spatial dimension (z-axis) and a temporal dimension. The CNN terminates with fully connected neural network which minimizes the loss function (cross-entropy in our case) over the training period, and outputs a probability that the patient has autism. The project is still ongoing, and the main obstacle is achieving the computational power to do our tests; we hope to have results soon.

Motivation:

Machine learning draws its origins from pattern recognition, computational learning theory, and artificial intelligence. The basic idea is for a program to be able to learn from its environment and solve problems without being explicitly programmed to do so. The exact way this works is by examining inputs and drawing comparisons between them in order to predict the result with a certain degree of probability, as opposed to solving the problem in a deterministic way. This process saves both time and resources when approaching certain problems in industry and science.

Machine learning is having more and more use in the modern world. In the last decade, several fields have experienced a humongous growth in the volume, complexity, and lack of clarity of simulated and measured results. An example of this can be seen in astronomy. Panoramic digital detectors cause the amount of processed synaptic sky curves to double every 12-18 months. Now, the main issue that arises is the inability to process this huge amount of data and draw valid conclusions in a realistic amount of time. This is where machine learning comes in. Using Google's powerful TensorFlow library, we will try to solve these emerging problems in industry using machine learning techniques.

To accommodate for the increasing complexity of problems, our techniques and algorithms also have to increase in capability. Image classification is a simple example of a problem that does not scale well for traditional machine learning techniques. As we increase the spatial dimensions as well as the number of input channels, the time to process the image in standard ways increases drastically. Another drawback is the excessive memory required to process the image, imposing a new requirement on the machines that are doing the work. As a

response to this problem, convolutional neural networks have gained much popularity as they offer the same amount of accuracy with exponentially less processing time and memory usage. If these networks did so well in bypassing the problem of high – dimensionality when it comes to image recognition, then we could expect that they could perhaps help with a much wider set of problems. This prompted our research into further applications of convolutional neural networks, and whether we can direct them to classifying data with considerably higher dimensionality.

Convolutional Neural Networks:

The generic neural network has been in use for several decades, and convolutional neural networks build off of their predecessor. The fundamental unit of these networks, the neuron/perceptron, remains unchanged. It receives inputs which can be thought of as a vector $(x_0, x_1, x_2 \ldots)$ and computes the dot product with the weight vector $(w_0, w_1, w_2 \ldots)$ and saves this dot product in a number, call it A. We then apply a non-linearity to A (several to choose from), and save it into B. Then, B is returned in the form of the "firing rate" of the neuron/perceptron.

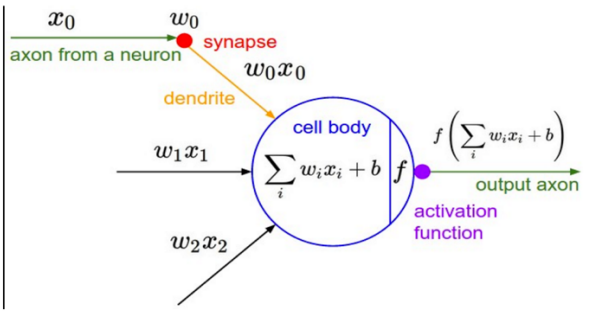

Figure 1

Convolutional networks are specifically made for image recognition. An image with dimensions $H\ x\ W\ x\ C$ would need that number of weights per neuron, causing the traditional model to be slow and bulky, not to mention possibly over fit the data.

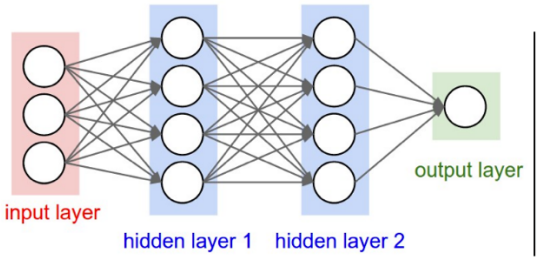

Figure 2

Convolutional Neural Networks take advantage of the fact that input can be thought of as 3 dimensional inputs. (height, width, color channels). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the

entire depth of the input volume. The use of filters and partial connectivity from one layer to the next allow the same if not better performance while being exponentially more efficient.

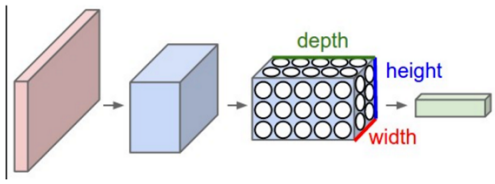

Figure 3

The 5 layers that make up convolutional networks are shown below in *Figure 4*. Each layer can be repeated as many times depending on the specific network and what we are trying to accomplish. For example, if our data is especially large, we would want to have several pooling layers. Another example is if that if our network is especially long, we would want two fully connected layers so that the correlations between different parts of the data is not lost in the process.

- INPUT Layer: [H x W x C] with raw pixel values.
- CONV Layer: Consists of filters that compute the convolution of the image with a learned weight vector.
- RELU Layer: Element - wise activation function, such as the max(0,x) thresholding at zero.
- POOL layer: Down sampling operation along the spatial dimensions.
- FULLY CONNECTED layer: Compute the class scores and return the probabilities for classification for each potential class. This is where we have our loss function and where we are minimizing it.

Figure 4

Initial Tests:

We did some baseline tests on the convolutional neural networks before we moved on to the actual project. The first data set we experimented with was the Iris Data Set. There are 150 data points, each of which are 4 - dimensional vectors, and a corresponding class which corresponds to the type of flower that it is. We experimented with both convolutional neural networks and generic deep neural networks to see baseline results. Below are the results of the deep neural network depicted in *Figure 5, 6, 7.* Our testing sample was a set of 30 flowers, and we indicated the probability of classification to each class. The color of red indicates the predicted class, and the color blue (if present) indicates the actual class (in the case where predicted class and actual class don't match).

Sample	Class 1	Class 2	Class 3	Correct Class
1	0.00255513	0.963146	0.0342992	1
2	6.48E-06	0.184529	0.815465	2
3	0.992481	0.0075186	1.13E-09	0
4	0.00165332	0.963993	0.0343533	1
5	0.00277461	0.918272	0.0789532	1
6	0.00380918	0.982331	0.01386	1
7	0.998071	0.00192946	5.75E-12	0
8	0.00012001	0.551666	0.448214	2
9	0.00522768	0.977295	0.0174774	1
10	2.44E-07	0.0207774	0.979222	2
11	3.47E-07	0.0387548	0.961245	2
12	0.992982	0.00701811	1.42E-09	0
13	2.34E-06	0.0456054	0.954392	2
14	0.00188548	0.852341	0.145773	1
15	0.00647622	0.975394	0.0181302	1
16	0.996285	0.00371502	9.37E-11	0
17	0.0422964	0.95599	0.00171316	1
18	0.994022	0.00597789	7.13E-10	0
19	0.995915	0.00408537	1.41E-10	0
20	6.72E-07	0.0203202	0.979679	2
21	0.996408	0.00359225	8.12E-11	0
22	0.00080587	0.827074	0.172121	1
23	1.56E-06	0.0597539	0.940245	2
24	0.00011481	0.482809	0.517077	1
25	0.00886843	0.981563	0.00956857	1
26	0.00130325	0.940863	0.0578341	1
27	0.997549	0.00245088	1.59E-11	0
28	0.00128333	0.971974	0.0267427	1
29	7.31E-08	0.0134275	0.986572	2
30	0.00433767	0.986691	0.00897091	1

Figure 5

	Actual Class 1	Actual Class 2	Actual Class 3
Predicted Class 1	8	0	0
Predicted Class 2	0	14	0
Predicted Class 3	0	1	17

Figure 6

```
Accuracy: 0.933333
```

Figure 7

Now below are the results from the convolutional neural network.

Sample	Class 1	Class 2	Class 3	Correct Class
1	0.371671	0.128967	0.499363	1
2	0.347279	0.124356	0.528366	2
3	0.451697	0.133066	0.415237	0
4	0.374397	0.126433	0.49917	1
5	0.367205	0.131802	0.500992	1
6	0.367615	0.127403	0.504982	1
7	0.490878	0.126206	0.382916	0
8	0.352531	0.124637	0.522832	2
9	0.377021	0.128422	0.494557	1
10	0.332063	0.124131	0.543805	2
11	0.337877	0.121939	0.540184	2
12	0.489055	0.132413	0.378533	0
13	0.357668	0.128142	0.51419	2
14	0.362174	0.132888	0.504938	1
15	0.359206	0.13208	0.508714	1
16	0.47979	0.12973	0.39048	0
17	0.377297	0.132242	0.490461	1
18	0.463917	0.128534	0.407548	0
19	0.474379	0.130029	0.395592	0
20	0.348093	0.129438	0.522469	2
21	0.4654	0.128958	0.405641	0
22	0.369799	0.127585	0.502616	1
23	0.346053	0.12286	0.531087	2
24	0.347834	0.127693	0.524473	1
25	0.37169	0.130374	0.497936	1
26	0.365526	0.126312	0.508163	1
27	0.492564	0.125909	0.381527	0
28	0.362047	0.124957	0.512996	1
29	0.347554	0.123151	0.529295	2
30	0.366439	0.127011	0.506551	1

Figure 8

	Actual Class 1	Actual Class 2	Actual Class 3
Predicted Class 1	8	0	0
Predicted Class 2	0	0	0
Predicted Class 3	0	14	8

Figure 9

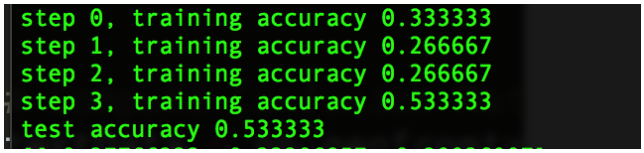

Figure 10

Surprisingly, our DNN had a much higher accuracy than our CNN. We can attribute this to the fact that the small amount of features in our data set don't work well with the CNN methodology. In this case obviously a normal fully connected neural network works much better. So, we decided to do baseline tests of the CNN on a different data set that is more suited to having local spatial patterns. Specifically, we tested on the MNIST data set, which is a dataset of handwritten digits. Using a sample of over 100,000 of these images, we were able to achieve an accuracy of 99.2%.

These initial tests verified that the convolutional neural networks can achieve very high rates of accuracy, but they also showed the limitations of these types of networks. They seem to work best for tasks where patterns are found in local spatial sections, which would make sense as these patterns would trigger specific filters in the convolutional layer. This guided the rest of the project as we realized that any high dimensional dataset we used would have to have these local spatial connections in order for the convolutional neural networks to work well.

The Problem:

We now turned our attention to the problem of video recognition, which we deemed to be a sufficiently complex problem to test our convolutional neural networks on. In addition, this could be a pathway into using the convolutional networks to other high dimensional data sets, as it serves as a good midway point between simple image recognition and something as complex as classification of celestial objects.

Specifically, we turned to the problem of diagnosing autism using nothing but fMRI clips. The public dataset came from the ABIDE, or Autism Brain Imaging Data Exchange, organization, and was from a study involving around 150 patients at the University of Maryland. The dataset is relatively even in the amount of patients with autism, and without. Each fMRI clip is a 64 x 64 x 40 x 300 clip. The first three dimensions are spatial, and the last is temporal. These images are all grayscale images, and examples can be seen below of a random patient at a random time segment at different depths. In essence, each patient's fMRI clip is a matrix of rank 4 with nearly 50,000,000 entries.

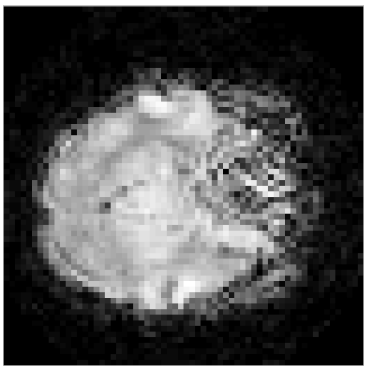

Figure 11

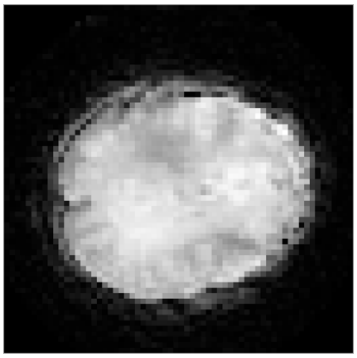

Figure 12

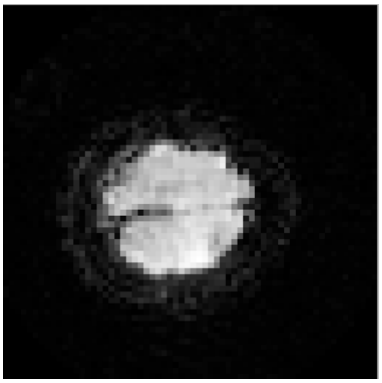

Figure 13

An immediate challenge that is presented here is that in a classic image classification problem, although the images are three dimensional, the third dimension was color channels and not depth. This would mean that we would have to reshape all of our clips into matrices of dimension 64 x 64 x 1 x 40 x 300 instead for better results. There are several problems with this however. First of all, reshaping a matrix with near 50,000,000 entries is not a task that a computer can do in a fast amount of time. Even with clever manipulations of list slicing and re-declarations of portions of the matrix in Python, this is still a lengthy process. Second, TensorFlow does not support convolution and pooling operations over input data that is 5 - D. This problem is not as troublesome though, as we can write custom functions as the work progresses. Another problem we encountered is the order of the dimensions of the data. In NIFTI format, the first three dimensions are spatial and the last is temporal. However, for a convolutional network, filters extend through the depth of the input (the last dimension) and are

through a small spatial size. This means that we need our input to actually be in the form 300 x 64 x 64 x 1 x 40. Here again we encounter the problem of reshaping a matrix with 50,000,000 entries, which is no easy problem.

Our attempts at the full scale version with reformatted data, as well as a full convolutional neural network, were far too computationally intensive for the machines we had at our disposal. Instead, we decided to work with smaller networks, and entirely skipped the step of reformatting the data and saved that task for later when we had better resources. The first network we used is shown below, and the architecture of the network has been drawn out for ease of understanding.

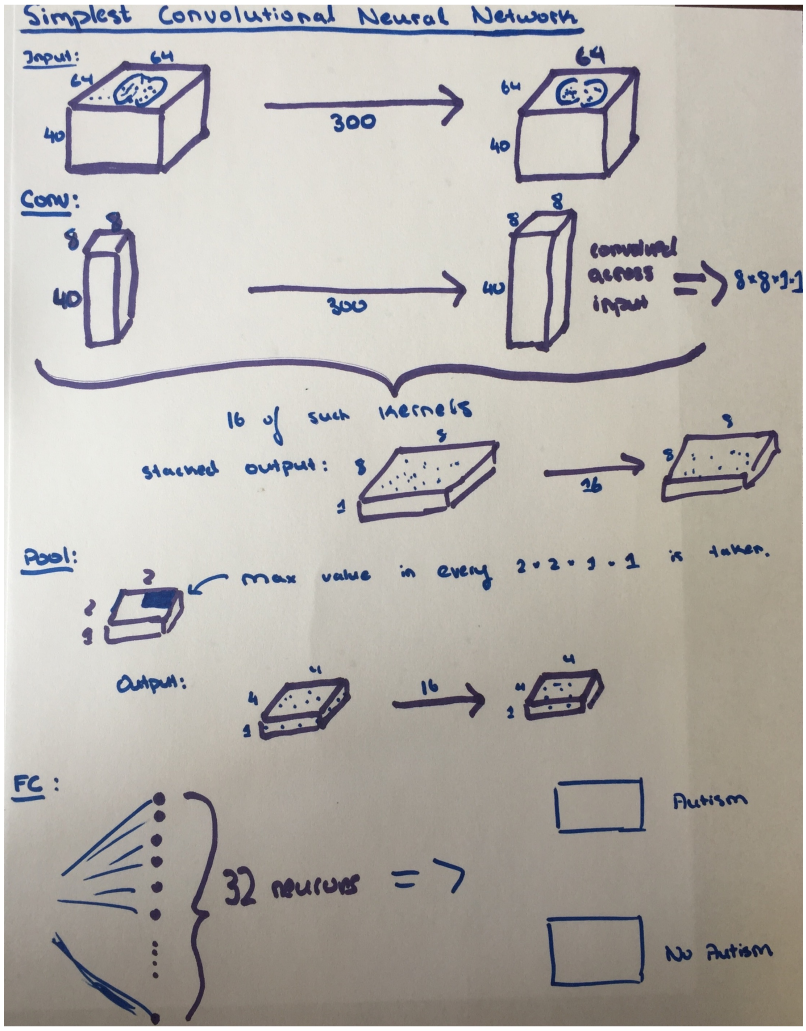

Figure 14

We split our data of 150 patients into 24 training batches with 5 clips each, and a test batch with 30 clips. The entire time to train, and test the network was a little over 4 hours with our resources. This network was modified in small ways (but general architecture was maintained) and over the average of different trials, the mean accuracy was 53%. Desiring more accurate results, the network was upgraded, as can be seen below.

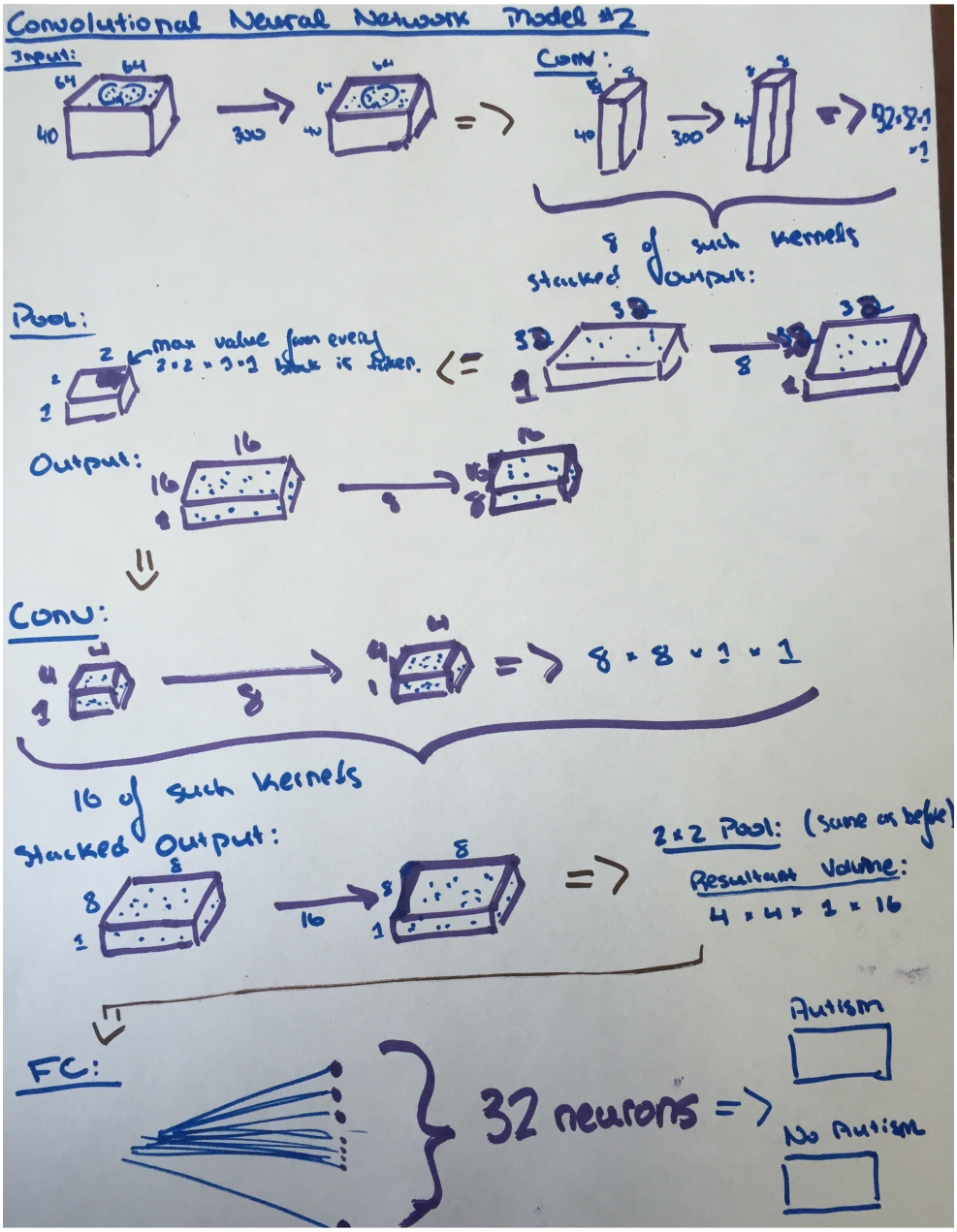

Figure 15

The entire time to train, and test this network was a little over 14 hours with our resources. This network was modified in small ways (but general architecture was maintained) and over the average of different trials, the mean accuracy was 56%, and one of the runs had a high of 59%.

Conclusion:

It is very important to emphasize the relative simplicity of all the convolutional networks we used. The low accuracy rate reflects this. The ideal network we wanted to use involves

i) reformatting of our data as previously described ii) two fully connected architectures instead of one at the end of the network iii) using a much high number of neurons in every single layer of the network. An exact description can be seen below.

```
# IDEAL GOAL NETWORK

# INPUT VOLUME = (300, 64, 64, 40)
# CONV LAYER = (10, 32, 32, 40, 64)
#   output of this will be (30, 32, 32, 64)
# POOL LAYER
#   output of this will be (30, 16, 16, 64)
# RELU LAYER.
# CONV LAYER = (10, 8, 8, 64, 128)
#   output of this will be (3, 8, 8, 128)
# POOL LAYER
#   output of this layer will be (3, 4, 4, 128)
# REUL LAYER
# CONV LAYER = (3, 2, 2, 128, 256)
#   output of this layer will be (1, 2, 2, 256)
# FC LAYER 4096 Neurons.
# FC Layer 4096 Neurons.
# BINARY OUTPUT
```

Attempts to run this network have been unfruitful as the machines we use lack the sufficient RAM, and even if they did, the training of the network itself would take over 150 hours by our estimations. The only way to make further progress is to be able to acquire sufficiently power computers to be able to hold all of the data at once as well as do the computations quickly. Using several servers is one of several solutions to this problem. However, it is a good sign that our accuracy increases as we increase the complexity of the network. We can only hypothesize that our goal network would thereby give us a much higher accuracy.

Although we did not definitively answer the question of whether convolutional neural networks serve as a good way to classify data in high-dimensional datasets, we have set the groundwork for future research on the topic. We have shown that these networks require a tremendous amount of memory and computational power, but perhaps this will result in a significantly higher accuracy rate compared to current techniques. Hopefully future research will be able to build off these initial results and reach a more conclusive answer.

References:

1. Fausett, L. V. (1994). *Fundamentals of neural networks: Architectures, algorithms, and applications*. Englewood Cliffs, NJ: Prentice-Hall.
2. A. (n.d.). *ABIDE_UM* [NIFTI].
3. CS231n Convolutional Neural Networks for Visual Recognition. (n.d.). Retrieved August 14, 2016, from http://cs231n.github.io/convolutional-networks/
4. Convolutional Neural Networks (n.d.). Convolutional Neural Networks (LeNet)¶. Retrieved August 15, 2016, from http://deeplearning.net/tutorial/lenet.html

Acknowledgements:

I would like to thank my mentors, George Djorgovski and Ciro Donalek, as well as Paola Davis and Ashish Mahabal for all their help.