Convulotional Neural Network on Hand Written Text

Iyuehan Yang
Computer Science
Brock University
St Catharines, Ontario
Email: iy12qm@brocku.ca

Dan Usman
Computer Science and Businsess
Brock University
St Catharines, Ontario
Email: du11ph@brocku.ca

***Abstract*—The purpose of this program is to design, implement and test a convolutional neural network and determine whether or not it is able to correctly classify hand written text. The program will be written in Java using the Deep Learning 4J library in order to quickly create the neural net to test and determine the results. The results show that a convolutional neural network greatly exceeds the accuracy of recognizing handwritten letters compared to feed forward back propagation network. We were able to compare two different architectures of the network; AlexNet and LeNet. AlexNet concluded with overall better results. For extracting results between the two architecture, we utilized the DeepLearning4J library. The project was executed on Eclipse Neon Release 4.6.0 using Java 8.**

I. Introduction

This document is a report for our final project in the course COSC 4P80: Neural Networks. The goal of this project is to implement a convolutional neural network which is able to find letters on an image and correctly classify them to the appropriate letters. Currently, using a normal backpropagation feed forward neural net, the system is unable to correctly and accurately classify hand written letters on a page. The regular neural network is only able to accuratly classify them if the input contains only one letter centered on the image. Therefore, as seen in ImageNet, convolutional neural networks are able to accurately find and classify different objects in an image. We will be using that concept in order to find and classify hand written letters. The system will be integrated using the Deeplearning4J java libraries and trained on the dataset BLANK. We will empirically determine the appropriate parameters required in order to get the best results in our convolutional neural network.

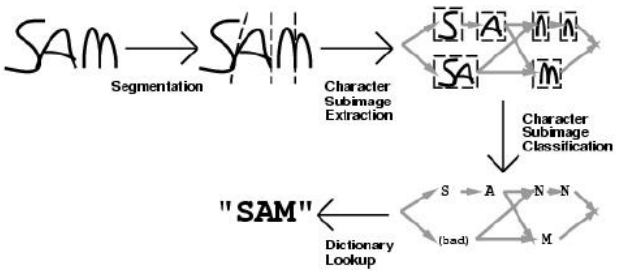

Fig. 1. Handwritten letters recognition

II. Convolutional Neural Network

A. Architecture Overview

In a regular neural network structure, the networks receive an input (a single vector) and transforms it through a series of hidden layers. Each hidden layer is then made up of a set of neurons where each neuron is fully connected to all the neurons in the previous layer. The last layer is called the output layer and is used as classification. Regular neural networks do not scale well to full images. For example, if an image was of size 32x32x3 (32 px wide, 32 high, and 3 color channels), a single fully connected neuron in the first hidden layer would have 32*32*3 = 3072 weights. As an image scales up to a higher quality, e.g. 400x400x3, that would have 400*400*3 = 640,000 weights. Clearly, this would lead to overfitting as the number of parameters become too large.

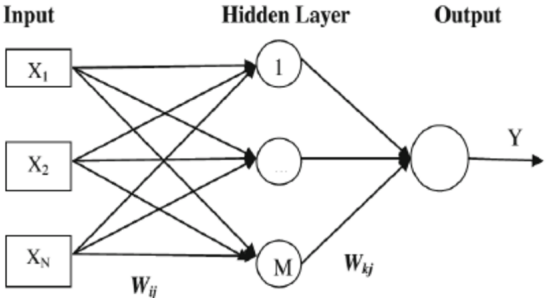

Fig. 2. Handwritten letters recognition

In machine learning, a convolutional neural network is a type of feed-forward artificial neural network where it's inspired by the organization of the animal's visual cortex. In essense, as animals see images, individual neurons respond to stimuli in a restricted region of space known as the receptive field. These fields will then partially overlap on top of each other in order to create the visual representation of what we see. This will be the bare bone of how convolutional neural networks are implemented and structured

A standard convolutional neural network has a sequence of layers, and every layer transforms one volume of activations

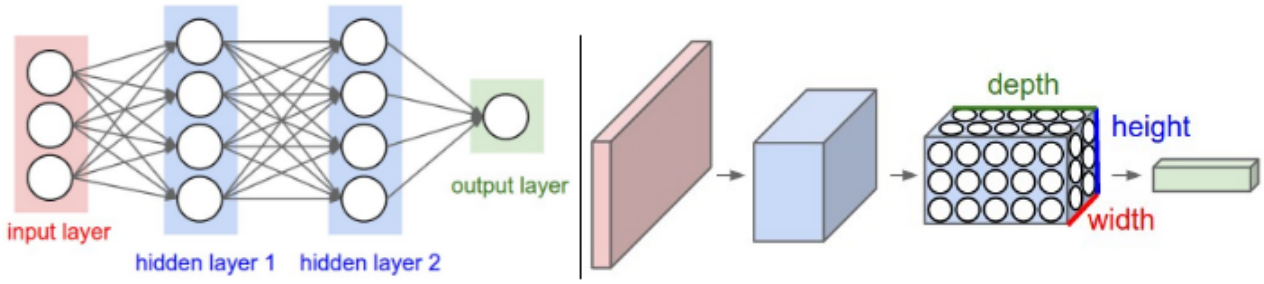

Fig. 3. Convolutional Neural network Architecture

to another through a differentiable function, just like a regular neural net. However, there will be an additional 2 layers which are used specifically for convolutional neural networks.

Example Archtitecture:

- INPUT [32x32x3] layer which holds the raw pixel values of the image. E.g. width = 32, height = 32, 3 color channels
- CONVOLUTIONAL [32x32xN]layer which computes the output of the neurons connected to the input layer. Each of the convolutional layers will do a dot product on a filter specified for each feature and a small region connected to the input layer. This results in a set of activation maps depending on the number of filters. E.g. [32x32x12] if we have 12 filters.
- RELU layer which is essentially an activation function.
- POOL layer which down samples the dimensions resulting in a smaller sized product.
- FULLYCONNECTED layer which is just like the regular neural networks and is used for classification of ur results.

Convolutional neural networks take advantage of the fact that the input consists only of images. Unlike a regular Neural Network, instead of being fully connected, the neurons in a layer will only be connected to a small region of the layer before it. The convolutional neural net will then transform the original image layer by layer from the original input attributes into a classficiation.

B. Convolutional Layer

The convolutional layer is the core building block of the network and is used to calculate and determine a set of features from the input.

Each convolutional layer will have a feature they are trying to detect from the image. The size of the filters will be a small subset of the input image. For example, it might have size 5x5x3 while the input image is 32x32x3. This filter will then slide across the input image and will compute the dot products between the entries of the filter and the input at all the positions. Essentially, it is using the sliding window technique and adding a score to each sub-window based on the dot product between the filter and that sub-window. Each filter will then produce a set of 2-dimensional activation map which will then be stacked together in order to create the output volume.

In the convolutional layer, there are a set of parameters which are used to alter its capabilities.

1) Number of Filters
2) Local Connectivity
3) Stride
4) Zero Padding

1) Local Connectivity: We saw that is it impractical when dealing with high-dimensional inputs such as images in a regular neural net. The fact that it is fully connected causes the network size to grow too large as the dimensionality increases. Therefore, instead of connecting all the neurons in one layer to the previous layer, we will only connect each neuron to a subset of the input layer. This is known as the receptive field of the neuron, which is equivalently the filter size and will be set based on the user parameters.

2) Stride: The stride must be specified and is used to determine how many pixels the sub-window will move when it's sliding across the input image. The larger the stride parameter, the smaller the ouput volumes and vice-versa.

3) Zero Padding: In some cases, it will be required to pad the input volume with zeros around the border and the zero padding parameter will determine how many zeroes will be added. The reason why we use zero padding is to control the spatial size of the output volumes in order to make it a consistent size as the input volume.

The spatial size of the output volume is computed as a function of the input volume size ***(W)***, the receptive field size ***(F)***, the stride ***(s)*** and the zero padding size ***(P)***. The formula is ***(W - F + 2P) / S + 1***

For example, for a 7x7 input and a 3x3 filter size with stride length 1 and pad size 0, we would get a 5x5 output. If we increase stride length to 2, we would get a 3x3 output. Now,

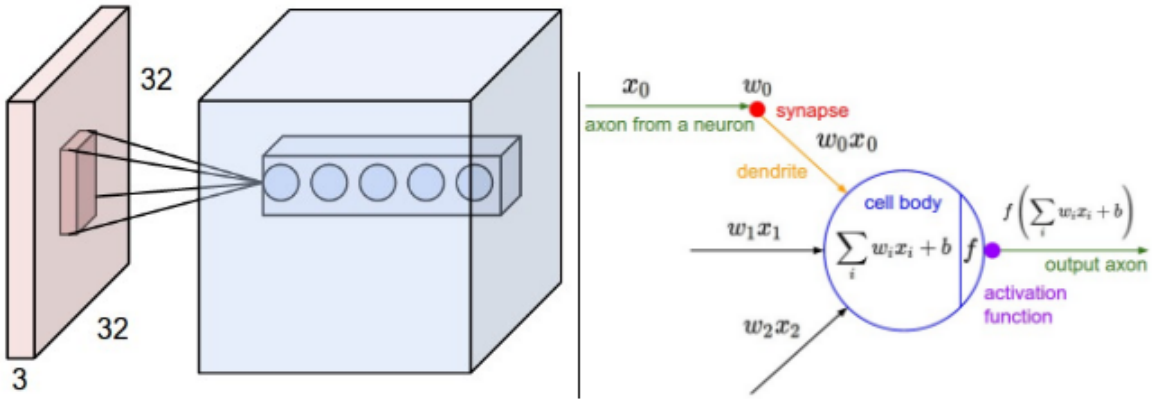

Fig. 4. Local Connectivitiy

if we were to set stride length to 3, the formula would produce a value of 2.33 which is not an integer. Thus we would have to increase the zero padding to 4 which will result in a 5x5 output.

Fig for example.

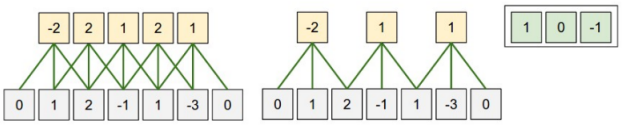

Fig. 5. Spatial Arrangement

4) Parameter Sharing: Parameter sharing is used in convolutional layers to control and reduce the total number of parameters used by making one reasonable assumption.

- If one feature is useful in some position (x, y), then it should also be useful at position (x2, y2). By denoting a single 2-dimensional slice of depth as a depth slice ([32x32x3] has 3 depth slizes of size [32x32] each), we can make the neurons in each slice use the same weights and bias.

5) Matrix Multiplication: Since the calculations in the convolutional layer performs many dot products between the filters and the sub-windows of the input, we can formulate the forward pass as one big matrix multiplication as follows:

1) For example, if the input is of size [227x227x3] and will be convolved with [11x11x3] filters at stride 4, then we take the [11x11x3] sub-window from the input and streach each block into a column vector of size 11*11*3 =363. Going through this, we would get 55 locations along both the width and height of the input image leading to an output matrix of size [365x3025], where every column is a stretched out sub-window and there are 55*55 = 3025 of them.

2) The weights are also similarly stretched out into rows.

3) The result of a convolution is then the same as performingone large matrix multiplication which willl evalute the dot producy between every filter and each sub-window.

4) The result must finally be reshapred back to its proper output dimension.

C. ReLU Layer

ReLU (Rectified Linear Units) is a layer which applies the activation function ***f(x) = max(0,x)***. This will increase the non-linear properties fo the decision function without affecting the receptive fields of the convolutional layer. The gradient computation is very simple - either 0 or 1 dependingon the sign of $\boldsymbol{x}$, which prevents the vanishing gradient problem.

D. Pooling Layer

The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters in the network, which directly reduces the computational time. Pooling is a form of non-linear down sampling and the most commonly used technique is called max pooling. Max pooling partitions the input image into a set of non-overlapping rectangles, and for each sub-region, outputs the maximum value of the feature set. The reasoning behind pooling is that we make the assumption that the exact location of a feature is less important than its rough location relative to other features.

However, in recent trends, alot of people are using less pooling layers or discarding the pooling layer altogether due to the fact that it is eliminating a lot of information. The alternative would be using a larger stride length in the convolutional layer once in a while or using smaller filters. In recent studies, discarding pooling layers have also been found to be important in training good generative models, such as variational autoencoders or geneative adversarial networks.

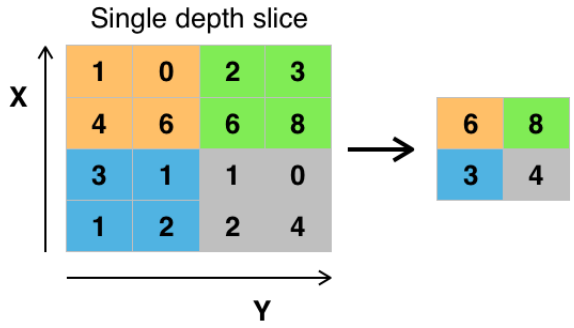

Fig. 6. Max-pooling with 2x2 filter and stride = 2

E. Fully-connected Layer

The fully connected layer have full connections to all activations in the previous layer, just like in a regular neural network. This layer is used as the end classification function to determine which class the input will represent.

F. Layer Patterns

Now that we know that convolutional neural networks are commonly made up of the 4 different layers - Convolutional layer, ReLU layer, Pooling layer and fully connected layer, we can decide how to use these different layers in creating a pattern which our network will use to be trained on.

There are many different ways you can create a pattern for the layers. The most commonly used one would be to stack a few Convulational - ReLU together and then reduce the dimensionality by applying a pooling layer. We will continue this trend until the image has been merged and reduced spatially to a small enough size. At some point, we will then transition this network to a full-connected layer, where it will train the high level reasoning behind the final classification. The full-connected layer will hold the final output.

1) Common architecture practices:

- Stacking multiple convolutional neural networks before a pooling layer is generally a good idea for larger and deeper networks. Stacking the convolutional layers before a pooling layer develops more complex and in depth features of the input before the destructive pooling. This way, the pooling will not cause the network to lose too much information

- A stack of smaller convolutional filters is preferred compared to one larger filter. Suppose we stack three 3x3 convolutional layers, each neuron in the first layer has a 3x3 view of the input. Each neuron in the second layer will then have a 3x3 view of the first convolutional layer which by extension have a 5x5 view of the input layer. If instead, this was switched to a single 7x7 convolutional layer, the neurons would be computing a linear function over the input, while the 3x3 stack of filters would contain non-linear attributes which makes the features more expressive.

- The input layer size should be divisible by 2 many times. Common numbers include 32, 64, 96, etc.

- In the pool layers, the most common setting is to use max-pooling with 2x2 receptive fields and with a stride of 2. This will discard exactly 75 percent of the activations in the input layer.
- Smaller strides in the convolutional layer works better in practice. This also allows the spatial down-sampling to be done in the pool layers while the convolutional layers will only focus on transforming the input.

III. Algorithm Used

In order to train our network, we used the DeepLearning4J library to create our convolutional neural network. DeepLearning4J is a java based neural network library which allows you to quickly set up a neural network in order to see results.

A. Models Used

AlexNet structure consists of 5 Convolutions Layers and 3 Fully Connected Layers. The first convolutional layer aims to transform the 3D input volume (height, width, color channels) to a 3D output volume by converting samples of 227x227x3 images to a 3D output of 55x55x96. The first layer concludes with 290,400 neurons with 363 weights for each neuron connected to the next layer, resulting in 105,705,600 parameters. Each convolution layer makes use of the RELU function, which has been said to converge 6 times faster than the tanh function when used on the CIFAR-10 dataset. The LeNet was the very first Convolutional Neural Network introduced. Its architecture consists of 2 Convolutions Layers with only one Fully Connected Layer. The input consists of 32x32 dimensions followed by an output of 28x28x6 and 14x14x6 in the following layers.

1) AlexNet Model: The AlexNet model consisted of 5 convolutional layers, 3 max-pooling layers, and 2 fully connected layers. The 5 convolutional layers used a filter size of 5x5 and 3x3. The model pooled the parameters after the first convolutional layer and then again after 3 consecutive 3x3 filter convolutional layers. The learning rate was set to 0.01, a decay rate of 0.1 and momemtum set at 0.9. The AlexNet model took longer to train due to have a more complex model compared to the LeNet model.

2) LeNet Model: The LeNet model is a much simpler implementation of the convolutional neural network. It had a total of 5 layers - 2 of which are convolutional layers, 2 pooling layers and a backprop to train the overall high level classification. The LeNet model also used a stochastic gradient descent optimization algorithm. The learning rate is set to a low 0.0001 with momemtum at 0.9. This model was a lot faster to train compared to the AlexNet model as it is simpler.

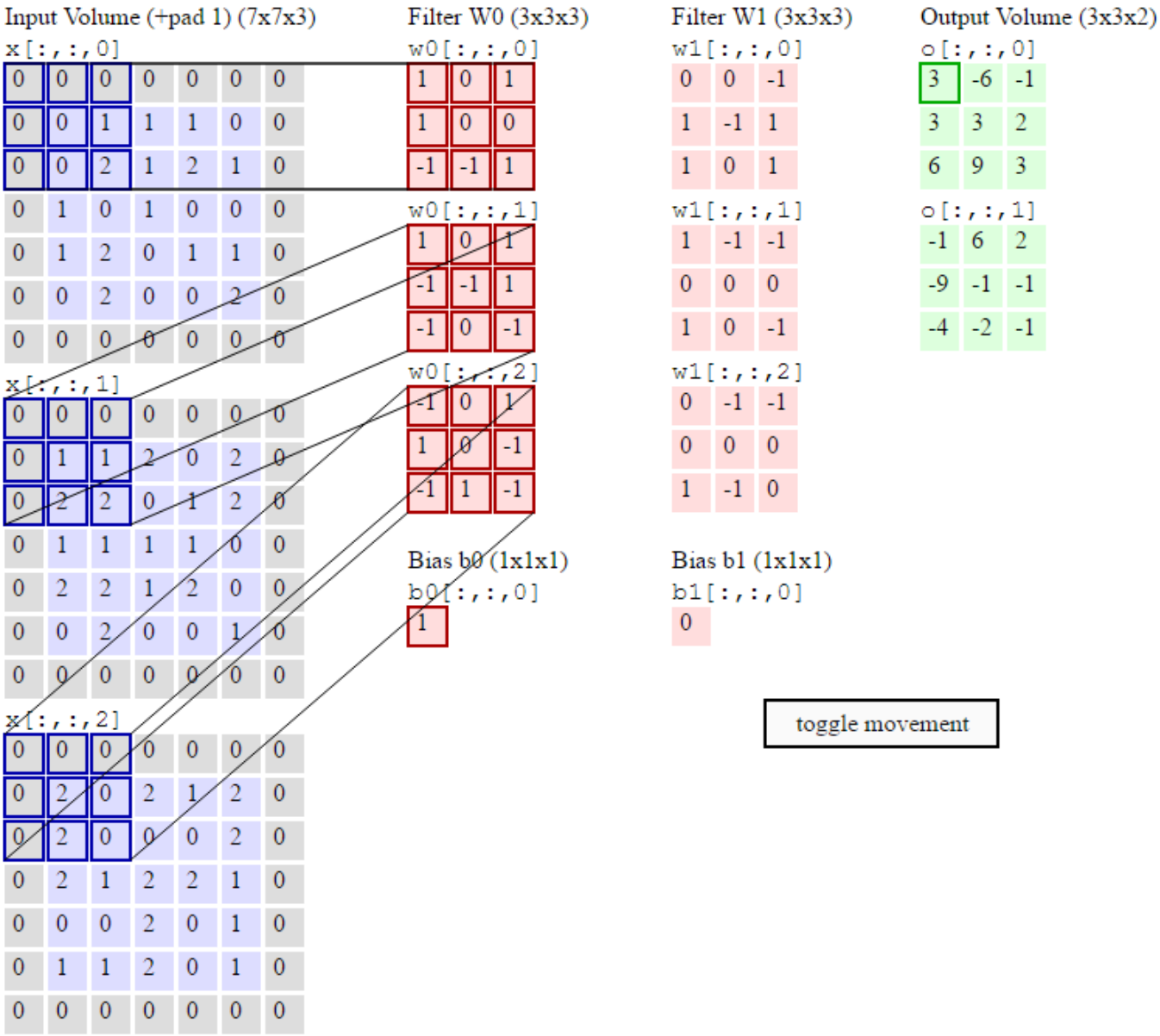

Fig. 7. Convolutional Layer Example - Computing the output volume

IV. Results

A. Result 1 (Fig 8)

- batch Size = 10
- Learning rate = 0.01
- Model = AlexNet

The network concluded with an accuracy of 80 percent on the testing data. It can be noted that as the network progresses through its epochs, there are fluctuation in the global error. This may be due to a great learning rate which causes the network to unnecessarily exploit the search space.

B. Result 2 (Fig 9)

- batch Size = 10
- Learning rate = 0.0001
- Model = LeNet

The LeNet architecture seems to provide us with a similar accuracy of 80 percent on un seen data. However, it can be noted that its a much smoother decent when compared to the AlexNet. It seems to suffer from the same over exploitation mentioned above. There are many parameters that can cause this fluctuation in the global error, however, we believe that it might be the learning rate.

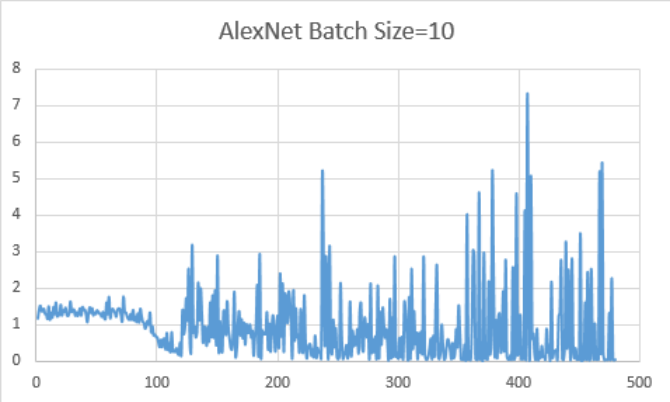

Fig. 8. AlexNet model - batch size = 10 - Learning Rate = 0.01

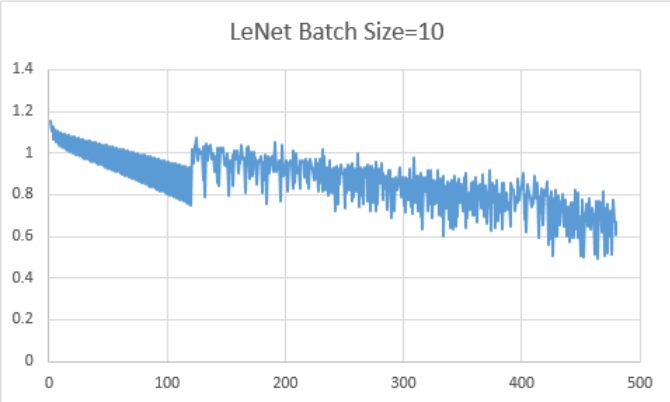

Fig. 9. LeNet model - batch size = 10 - Learning Rate = 0.0001

C. Result 3 (Fig 10)

- batch Size = 20
- Learning rate = 0.01
- Model = AlexNet

The LeNet architecture seems to provide us with a similar accuracy of 80 percent on un seen data. However, it can be noted that its a much smoother decent when compared to the AlexNet. It seems to suffer from the same over exploitation mentioned above. There are many parameters that can cause this fluctuation in the global error, however, we believe that it might be the learning rate.

D. Result 4 (Fig 11)

- batch Size = 20
- Learning rate = 0.0001
- Model = LeNet

An Increase in batch size produced a slightly reduced accuracy of 70 percent on the testing data. It can also be noticed that the results at each iteration follow exact same pattern as they did in Result 2, but it seems like there isnt a steep enough descent, but it does cause less fluctuations. There may be a correlation with the batch size and learning rate.

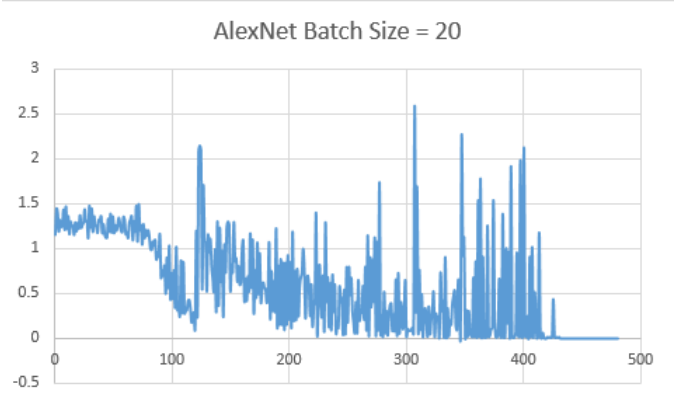

Fig. 10. AlexNet model - batch size = 20 - Learning Rate = 0.01

E. Result 5 (Fig 12)

- batch Size = 10
- Learning rate = 0.001
- Model = AlexNet

The decrease in learning rate while keeping optimal batch size proved to be disastrous and only proved to be 60 percent accurate on un seen data. It can be noted from the graph above that there isnt much progress being made towards the global minima. However, the magnitude of fluctuations is a lot less when compared to Result 1 and Result 3, with no sign of descent towards the global minima.

F. Result 6 (Fig 13)

- batch Size = 10
- Learning rate = 0.00007
- Model = LeNet

Provided with a lower learning rate, it didnt seem to affect the network by much. It managed to still provide us with at least 80 percent accuracy on the testing data. However, when compared to the previous results of the LeNet Architecture, it can be noted that it concludes with a steeper gradient descent than result 4, but not steeper than result 2. It also has a greater magnitude of fluctuation when compared to Result 4, but not as much Result 2. It seems like its the best of both worlds. You seem to get similar results when you increase the batch size or decrease the learning rate.

V. Conclusion

To conclude, due to the lack of data, our results may not be the most optimal. However, from what we gathered, the AlexNet model produced a higher accuracy compared to the LeNet. However, the LeNet model was faster to train. Throughout our experiments, we discovered that the batch size greatly affected the overall accuracy and the total time to train the network. We found out that this was the case for both the AlexNet and LeNet models. It can be noticed in Fig chart that it follows the same gradient descent as Fig

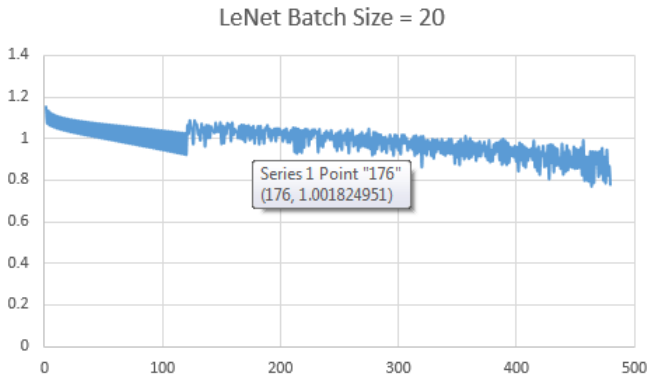

Fig. 11. LeNet model - batch size = 20 - Learning Rate = 0.0001

chart2, however it just took a longer time. A batch of size 10 was found to be optimal for both of the models. We believe this is due to the empiracally determined learning rate and momemtum which is set for both of the models.

In the future, we will train this convolutional neural network with the whole alphabet and see if the convolutional neural network would be able to correctly classify them all.

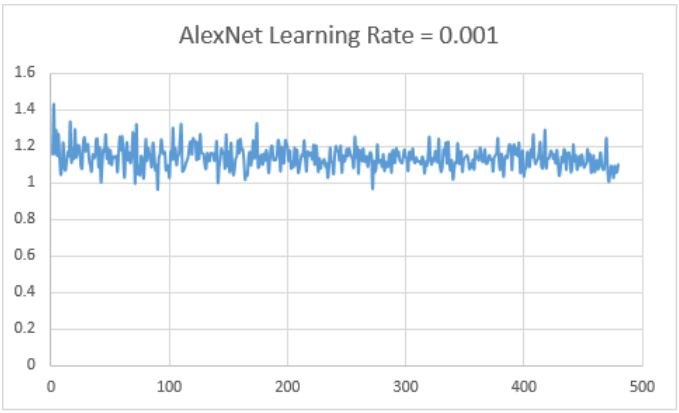

Fig. 12. AlexNet model - batch size = 10 - Learning Rate = 0.001

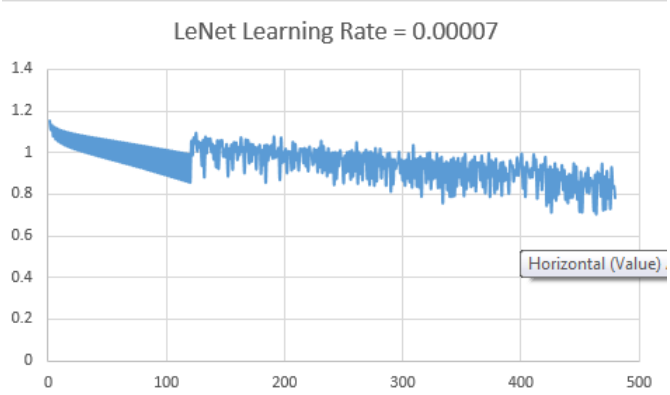

Fig. 13. LeNet model - batch size = 10 - Learning Rate = 0.00007

REFERENCES

[1] CS231n Convolutional Neural Networks for Visual Recognition, Cs231n.github.io, 2017. [Online]. Available: http://cs231n.github.io/convolutional-networks/case. [Accessed: 28- Apr- 2017].

[2] 2017. [Online]. Available: http://vision.stanford.edu/teaching/cs231bspring1415/slides/ale [Accessed: 28- Apr- 2017].

[3] Convolutional neural network, En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Convolutionalneuralnetwork. [Accessed: 28- Apr- 2017].

[4] An Intuitive Explanation of Convolutional Neural Networks, the data science blog, 2017. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Accessed: 28- Apr- 2017].

[5] A. Chris Nicholson, "Deeplearning4j Documentation Site Map - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM", Deeplearning4j.org, 2017. [Online]. Available: https://deeplearning4j.org/documentation. [Accessed: 28- Apr- 2017].