Jeffrey Kerley

Jakcqc

10/13/2020

[**Technical Description**](#_qub7s77isrna) **2**

[**Introduction**](#_c3ggpph1z3cc) **2**

[**Design Choices- Classification**](#_gmsyu6gdahpk) **2**

[**Output Methods - Softmax and Negative Log Likelihood**](#_db1wexjz6mrz) **3**

[**Neural Network Module- 2-D Convolutional layers vs MLP**](#_mb6ev4xdnflg) **5**

[**MLP**](#_lfgu8bsar9kt) **5**

[**CNN**](#_u0wifq2vcq5f) **6**

[**Experiments and Results**](#_k5dfu2k3bj9d) **10**

[**Neural Network differences - MLP and CNN**](#_xi2yzqgk2163) **10**

[**Classification**](#_k2y2zxp9wisq) **11**

[**Output Data**](#_qr9g0y2owt4k) **11**

[**Confusion Matrix**](#_2biw281hyjyd) **12**

[**Accuracy**](#_x24w7lvwcj05) **13**

[**Network Issues**](#_jf6tqmlsupmk) **14**

[**Lessons Learned**](#_43z19kvzctek) **15**

# Technical Description

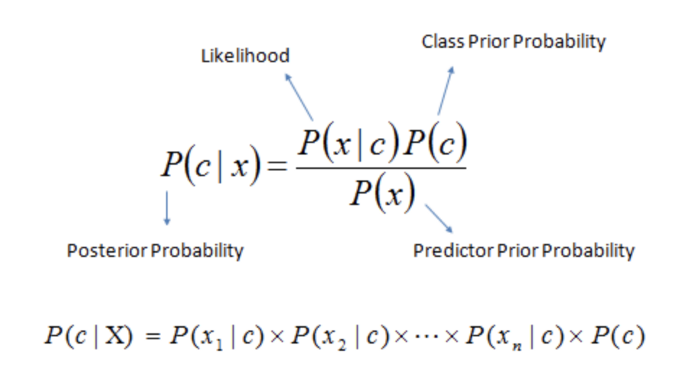
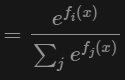
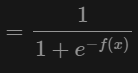
## Introduction

The MNIST data set is the basis for image classification practice. The reason for this baseline is that it is composed of 60,000 hand written images of numbers. Each of these are 28 x 28 pixels w/h, with no color besides black and white. The goal of this project is to implement a Convolutional neural network which will be trained to recognize these handwritten numbers, within the realm of acceptable accuracy. The basic implementation will follow as: loading the data, setting training information, creating the neural network module, optimization, training, testing, and finally, evaluation based on error to accuracy.

## Design Choices- Classification

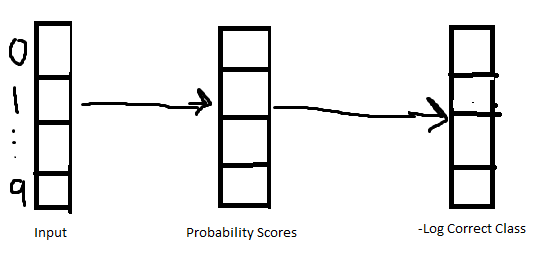
To first understand what needs to be built, the question of what is being asked needs to be understood. Classification of an image. A simple idea, but one that needs a lot of unpacking. The first problem with classification is how the classification is achieved. Binary versus multi-class, and in this case, multi-class would be necessary because of the 10 different digits which need classification. This provides multiple solution opportunities.

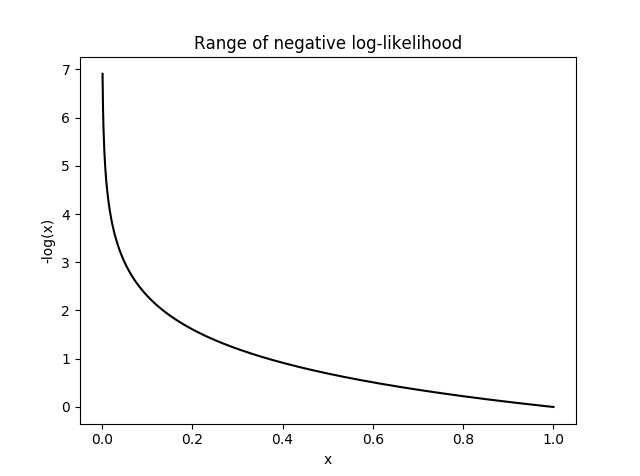
A decision tree is one solution. That is, for each of the data classes (0-9), rules are applied and a path is chosen down the tree based on features until the data finds the right leaf, in this case the right digit. The problems for this are simple with a dataset around images that do not have a clear cut definition and would require a complex tree. Instability and overfitting. The first solution to this would be to incorporate a random forest, multiple decision trees that use predictive accuracy and averages to fix the overfitting. Yet, the instability problem still stands.

Sklearn, Naive Bayes, confusion matrices, these are the tools that most people use when approaching the MNIST dataset, and for good reason. This method allows for easy and effective classification based on probability of previous observations. In this specific implementation of the neural network this was one of the classification methods. To the left the basic equation of applying class probability and getting a classification is shown. There are several different Naive Bayes models but in this case, Gaussian type is assumed (normal distribution). From this equation the derived formula gives the logistic function This is important because when applying this idea to multiple classes, 10 in this case, the result is a softmax-linear function defined as -----> Ultimately the purpose for this decision is to realize the data can be complicated and hard to model. It acts as a catch all and allows a simple convolutional neural network to be implemented for a complicated dataset. 

## 

## Output Methods - Softmax and Negative Log Likelihood



Above is how the softmax function is applied to the input array, onto the softmax probability scores, and then negative log-likelihood loss function. The range of this loss function looks as such: The goal of applying this is to get high probability numbers and to keep them that way. The predictor confidence value is thought of as high when probability is high, and vice versa. As a result of this a confusion matrix can be created to store our predicted, actual, and correct classifications. 

In this case, a 10 x 10 matrix is created storing the above data. Then using the expected outcome values, the predicted test dataset, and the calculated predictions and outcomes, the matrix is completed. This confusion matrix then evaluates the data on a basis of total accuracy, individual precision, and recall percentage of incorrect identifications. To further explain this matrix, each row represents the actual digit, so real 1, real 2, etc. Each column represents the predicted value of each digit, predicted 0, predicted 1, etc. As a result the [i][i] element is the number of accuracy predictions. [0][0] is all the correct guesses for 0, and [9][9] is all the correct guesses for 9. Knowing which classes are failing, which weights are being often predicted wrong allows for better adjustments for large data sets.

Importing the data from softmax to the confusion matrix is not the exact method, or reason to use one. Doing so in this manner is to provide better visualization and allow understanding on the test data after the training. Data displayed this way is easy to read and understand, and the more epochs run it gets cleaner and more concise. A figure[[1]](#footnote-0) shown in experiments will highlight this effect. Exact data in this case is still evaluated in terms of error and accuracy, but it uses two “matrices.” A confusion matrix and a softmax negative log-likelihood vector. Conceptually the softmax method made more sense, but visually the confusion matrix was effective, especially for debugging.

## Neural Network Module- 2-D Convolutional layers vs MLP

Although in project one part one, a design of a multilayer perceptron was done, a 2-D convolutional neural network is much more efficient with image classification. To use evidence, a test was run between a MLP and a CNN on the MNIST dataset.

### MLP

Two hidden layers of size 28 x 28 each, with 30 units total in each layer. Followed

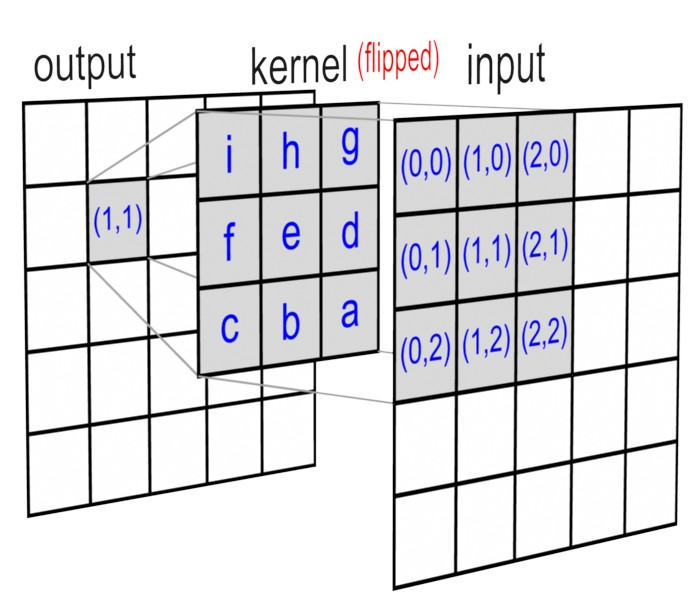
by a classification output layer with 10 classes for 0 - 9 digits. Log-softmax function is applied here for testing sake. The input from the MNIST dataset is flattened into a linear vector with a size of 784. 28 pixels \* 28 pixels = 784 pixels. Because there is no rgb this is very easy to flatten and normalize. The learning algorithm is Stochastic Gradient Descent with learning rate and momentum values of .01 and .5 respectively. After all of this was set up, the actual data is loaded which will be addressed in the data management section. This will cover the ins and outs of each different way to do so.

After the data is loaded, and trained within 30 epochs, the actual accuracy comes down to percentages around 97 percent. This is not optimized the most effectively but it is in the ballpark of a normal mlp. The average loss here was an amount lower than .08. Not to get into too much detail as project part one followed the in depth process of what was going on in each perceptron and algorithm, this is a solution to classifying images is no longer the best way. The problems that arise from this method will be addressed in the experiments and results section.

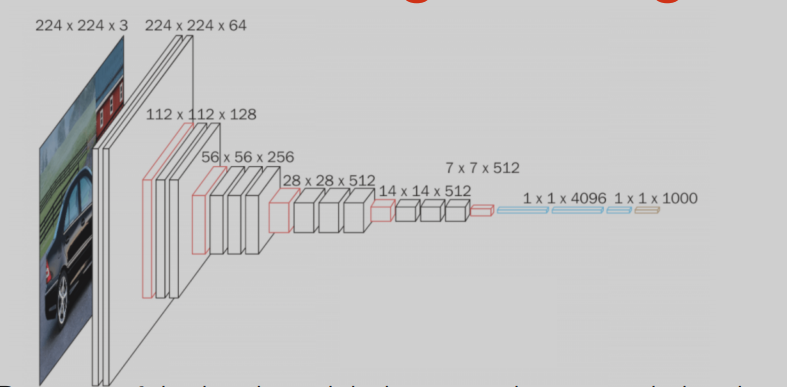
### CNN

The majority of time spent was on the implementation of a 2-D convolution neural

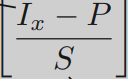
network. The basic idea here is breaking the layers into multiple filters to limit each layer’s inputs. Here: Two 2-D convolutional layers, with two hidden fully-connected linear layers, two dropout layers, pooling layers, and finally the output layer from our softmax/ReLU and confusion matrix. Dissecting the first layers: 2-D convolutional layers. These units are of the size 28x28 to fit the input data size. The caveat here is in the kernel. Described below , g(x,y) is the filtered image, and f(x,y) is the original. This is the basic expression for convolution. Each element is defined by and . 

With this basis, the kernel can be very powerful when applying the learning algorithm and optimization to each unit in the 2-d layer. In this case, a kernel of a 5 x 5 matrix is chosen. A good visualization of a 3x3 kernel applied on a 5x5 is seen below. Applying the kernel to each output, through each 2-d convolutional layer, we are able to effectively extract information from the data set (28 x 28) matrix into the next layer, and finally, to then send this information to be pooled, generalized, or sent to the activation function. 

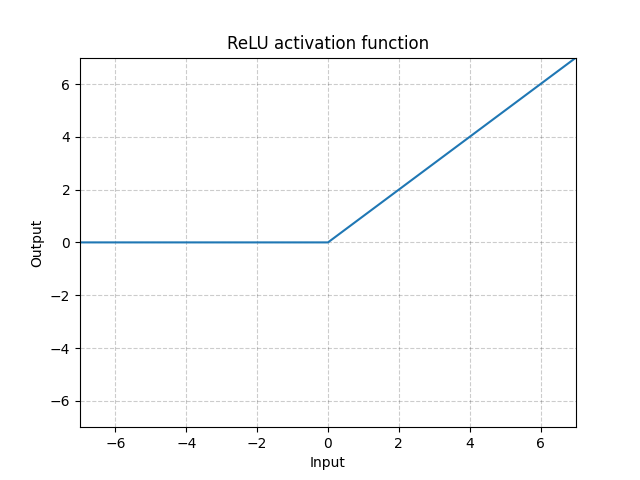
The goal is to grab the relevant information from the input and apply a learned filter to detected features. Essentially, this filter is how to get to the output of each layer. So, the first layer outputs its feature map, and the next layer receives this new data, and so on. This leads into the pooling layers. Max pool in this case applies another kernel which applies a “blur” in a way. That is, the information is less sharp to prevent small variations from changing the output significantly. Again, the effect is enhanced because the pooling acts as a way to break the information into subsamples. Achieved through limiting the inputs to each pooling layer max pooling reduces the total dimensions of an image.

Looking at the convolutional neural network on a global level the big picture is seen. 

To the left, the original 28x28 image will be broken into subsets and each layer applies different functionality to get the information to the activation function and output layer.

Conceptually the original data is not lost, it is applied in each of the layers, and the training of the network helps. This will lead into the other issue of having this many layers in a neural network: overfitting. The max function just takes the max of each quadrant in the input and makes a smaller output based on the max of the input. This is applied twice in this case. The equation follows as seen here, applies the equation: (image shape - Pool window size divided by the Stride, 2 in network, and the floor is taken). This roughly halves the input dimensions, keeping the max values from each section. 

Finally after each convolutional layer and pooling, the dropout layers are applied. Only applied on the training data, the purpose of this dropout layer just deactivates a certain amount of the correlations in the data. This is a percentage of data to be “ignored” to promote independence in the feature maps that are outputted. Certain dropout functions can be changed to simply slow learning rate instead of dropping the data, if the adjacent pixels are strongly correlated. Dropping an element means setting it to 0. The reason this prevents overfitting is because it keeps the neural network somewhat general, that is, it can still understand new data, while still learning the test data. If the model is overfit, the test data is known *too* well, and new data is unrecognizable. Essentially, max pooling, dropout layers, and convolutional layers all help keep a level of generalization in the network that would normally be uprooted through overfitting.

After all the convolutional layers, finally the activation function and optimizer can be reached. Rectified linear units, a ReLU is present in this neural network as the activation function. To apply the SGD with the backprop errors, a nonlinear function that resembles a linear function should be applied. Below is the graph of the function and what should be returned depending on the input. 

The equation: ReLU(x)=(x)+ =max(0,x) is applied to the output of the maxpool function. This function in unison with softmax, gives benefits of linearity optimization, while not infringing upon the multiclassed nature of the MNIST dataset. It activates when needed, and the softmax activation function will apply when it is needed.

This feeds into the gradient descent, error/loss, backprop, and optimizer which uses the learning rate and momentum to finally update the weights. As discussed extensively in project one part one, the equations and fundamentals behind these functions have been defined clearly, but will be explained briefly. The negative log-likelihood here computes the loss between the output and expected results. After this the back propagation is applied which collects the new gradients to apply onto the network. This is when the updates are applied as the training progresses to the weights.

One step that was not addressed, was the two fully connected linear layers. These neurons apply the updated weights to the incoming input data. The equation here is simple: y= xA^T + b. This is a basic linear transformation, which adds a bias(b), and uses the linear weight which goes on each edge. These are the principles which allow the neural network to learn.

The first step, which was left till the end top address was the actual data. This was done to keep some of the topics more abstract and more general, just as a neural network would do. The actual data is a set of images and labels. The labels simply allow the matching and subsequent matching of the predictions. They can be loaded in a multitude of different ways. A mean vector is a standard way to do this. As discussed the 784 size array is simple and effective. Another method would be to normalize and transform the data using the MNIST standard deviation and global mean. This is where knowing Gaussian distribution can save time and allow a better neural network.

# Experiments and Results

## Neural Network differences - MLP and CNN

The first Discussion surrounding the implementation of this neuron is the pitfalls of a MLP. Efficiency surrounding the MLP is no longer the best way to address classification problems. Filters, classes, cross-fold validation, and the effects of more hidden layers will still be discussed and the implementation will be dissected. The main focus, however, will be that of the Convolutional neural network and the experimentation and going into depth of implementing some of the same functionality that an MLP created.

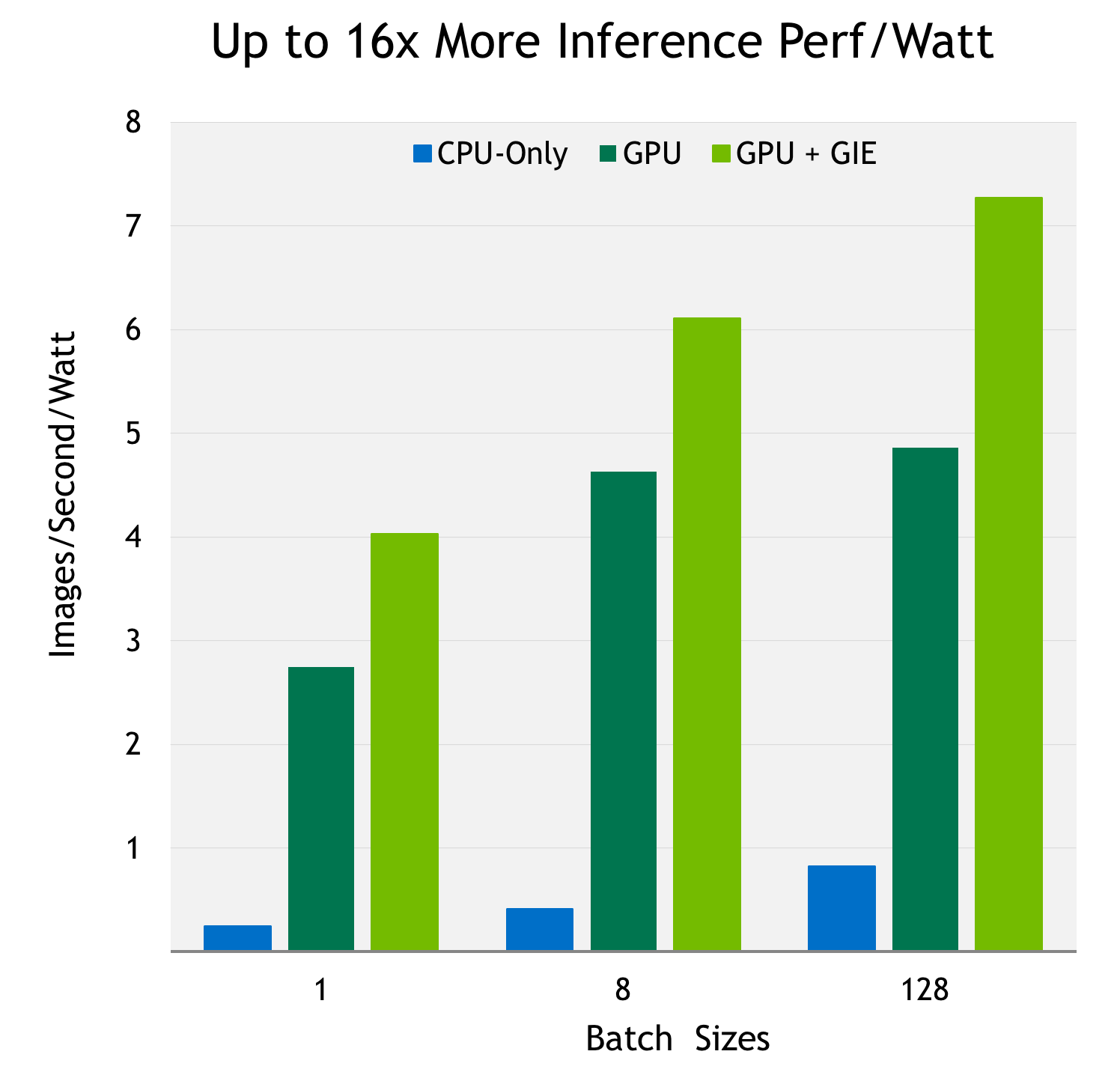
“The point of Part 2 is to have you apply a NN to data and *enjoy* the ups and downs of getting it to work, understanding whats going on, trying to decide what is a good solution and what is not, and evaluating your work.” This would include the decision of which neural network methodology to apply, in this case, a fully connected, big filter, two class problem. The two class problem responds to the fact that either it is the base truth diogiot, or it is not the base truth digit. Ten classes, as posed in the graduate level classes respond to the 10 classes of the digits, the functionality that a convolutional neural network imposes.

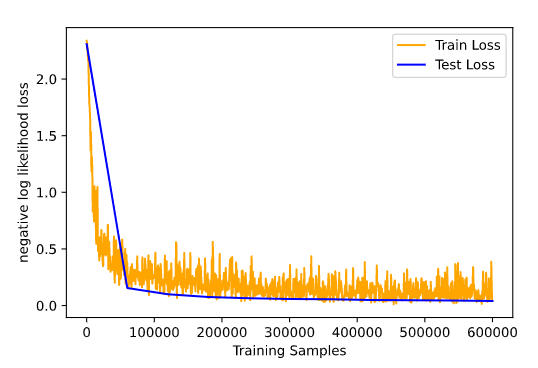
Addressing the validity of the decisions to work on part two this way is relevant in the scale of what is being asked and the abstract nature of the question. To begin understanding the changes, implementation challenges, and struggles of implementing this neural network, first the very concept of classification.

## Classification

The many different forms of classification needed to be addressed and as shown above there are a lot. Naive bayes, decision trees, logarithmic regression, and many more. I toyed with understanding each of these concepts before implementing open. I settled using a compilation of gaussian Naive bayes in respect to softmax. The figure below shows the struggle that was being understood and developed upon as the creation of the neural network was created. These were concepts which did not come easily, and took some real data to understand. The first iteration of my neural network did not use max pools, dropouts, and even applied certain activation functions too often.

## Output Data

My computer is running an AMD Ryzen 5, 16 gbs of memory, and a GTX 1080 as the GPU with 16 GB of memory. I set up an environment using CUDA to utilize my gpu to train data and actually log how the training would progress. Although this is not a part of the requirements It turned out to be interesting, and is worth the discussion. Nviodioa offers its own graphical interface to speed up performance and opn my specific machine IO found success on larger batch size and was able to do upwards of 30 epoch training in a short amount of time. This allowed more versions and teestiongs in a shorter amount of time. 



The largest part of my endeavor was working through the loss values, and evaluating on the accuracy, error, loss, and test loss. The below graph showcases how my neural network responds to multiple epochs (10), and the result in loss between training and testing.

The difference, Test set--> Average loss: 2.3069\_\_Accuracy: 1019/10000 (10.1900%).

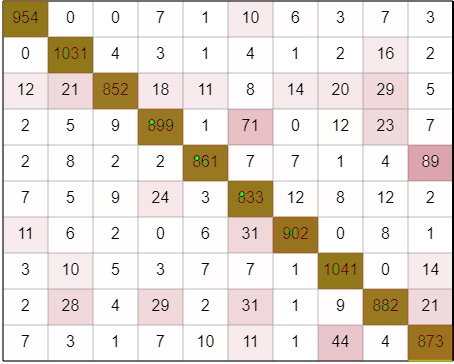
After the first epoch, Test set--> Avg. loss: 0.1560\_\_Accuracy: 9532/10000 (95.3200%).

And after 10 epochs, Test set: Avg. loss: 0.0408, Accuracy: 9877/10000 (98.7700%).

This was after utilizing maxpool, dropouts, and having every function running correctly. One of the reasons for using a convolutional neural network is for the extremely small loss value.

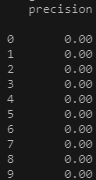
## Confusion Matrix

Next up was evaluating the confusion matrix. The confusion matrix below is based on the iterationV.1 of the MLP elimination which utilized sklearn for the normalization and math for it.



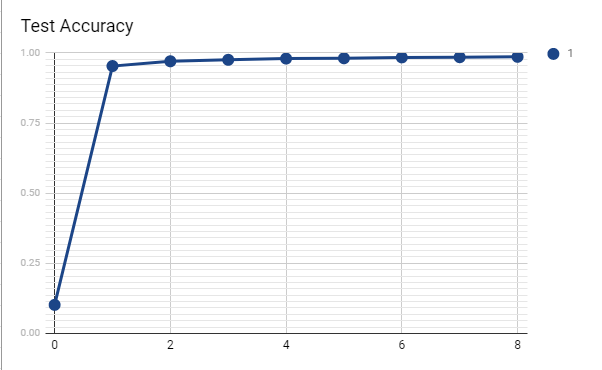
Precision below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| .95 | .92 | .96 | .91 | .95 | .82 | .95 | .91 | .90 | .86 |

Understanding this data is extremely straightforward and it is why confusion matrices are so helpful. Early in the creation of the MLP, I subdivided the data into chops of 300 images to save on time, and the use of the confusion matrix clearly showed the errors I was having. In that my precision values were way off. Well they were 0, which was a result of an incorrect precision statement in my math, even though the other values still remained correct. It is imperative that a clear visualization is present to debug in a neural network. Because of all the moving parts, even using pytorch, a lot can go wrong, and syntax errors are not always caught. I found this to be extremely tedious and was grateful to have set up the matrix when I did. 

## Accuracy

This segways into the idea of accuracy versus error plots. Pulling from the test accuracy lets see how the actual data progresses over 8 epochs on a simple graph. On the next page the graph justifies the neural network and showcases how the accuracy improves over time, landing at 98.6 percent accuracy! That is pretty good for a simple neural network! Although there are a lot of graphs above, It is necessary to understand how my neural network is actually learning. It is applying the changes, it is learning, and it doing so using all the above mentioned ways successfully.

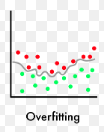


The evaluation is in accordance with proper learning techniques. Whether that be confusion matrices, error values, or even the accuracy of the neural network after each epoch.

It works!

## Network Issues

The question that still haunts this network, however, is how would more layers affect the network? Easy, overfitting and bottleneck are the perfect terms to describe what can happen and what solutions will arise.

Overfitting can easily be described as too many parameters and the network will find an answer when it really should not. I.E - the line of best fit does not generalize enough, which can result in test data being mislabeled or not understood. Bottleneck helps prevent this by including a layer that restricts parameters. So, more inputs than outputs. In the case of my neural network, all of the convolutional layers and maxpool actually are a form of bottleneck, in that they restrict the parameters, which prevents overfitting in one method. There was no individual bottleneck layer in this case, but the idea was transposed upon my individual layers, no need to create more than what is needed. 

Next up: the problem of too many parameters comes from an abundance of neurons. If there is too much power in the hidden layers, the test data can not actually train all of the neurons in the hidden layers. I went with two fully connected layers because it would be sufficient for the dataset at hand. This was not an issue for me, and when experimenting I never encountered any issues, but still kept it at two for simplicity.

Another decision was implementing the size of the kernels, size of the convolutional network input and output sizes, and applying the max pool size. Based on the systems in place for 2-D convolution, my kernel size has to be within a certain size to fit the input. In my case, a 3x3 kernel will actually not fit the input size. Same goes for a 6x6. So my options were a 4x4 and a 5x5. The 5x5 kernel resulted in an avg loss of .156 on one epoch and an accuracy of 95.3%. Whereas the 4x4 gave me an avg loss of .1681 and an accuracy of 95.02%. It is a slight difference but one that can add up over time to be significant. The 4x4 kernel gives a 98.09% accuracy, and the 5x5 kernel gives a 98.6% accuracy. The loss avg stays higher as well.

# Lessons Learned

I started this project with what I thought was a good understanding of neural networks and implementation. But as I told myself as I worked more and more, the day became a week of work. I realized that each topic I thought I understood was actually 10 times as deep as I realized. Pandora’s box would be a good analogy here. I was forced to ask the questions of why, and how, and then tasked with the challenge of showing all of my work, showing my changes, and explaining every last detail. I had to know what I was doing and saying. I feel very confident in implementing a neural network, especially on image classification, for a different dataset.

My journey took up about 70 percent of my computer's ram in chrome tabs, learning equations, functions, documentation, learning an entire new programming language, and understanding on a basic level what each moving part of my neural network was doing. I did get carried away making small changes that were not needed, but I was gripped and needed to make something I was proud of. I am not sure I was able to effectively showcase everything that needed to be shown, but I tried my best and am proud of my work.

I really thought that this part would be much easier than part one, given that sometime I struggle with the math notation and actual notation, but part two really displayed what information I was missing about neural networks. The classifications, the activation functions, the entire setup. I had to choose and choose correctly. Each of these topics could be studied for a semester, but I needed to grasp them in a couple weeks.

The things that did not turn out how I thought they would all correspond to how I implemented the network. Although I did create a simple MLP, I could not seem to understand the wide variety of experiments I was being asked to perform on it. I never expected to spend an entire day learning about kernels, and image processing (I am not in digital image processing either), yet, the idea clicked and I went with it. The ups and downs of a neural network seem to stem from all of the possibilities at the disposal. All of the choices that can be made. There is no “right” way to do it. If you get the right result that is what is proper, but it is achieved in countless ways.

I was also surprised by the history I started to uncover. The evolution of neural networks was unfolding right in my eyes as I learned old methods, functions, implementations, journal articles talking about new discoveries. Magical is an overstatement, but it did remind me of discovering programming. I really can not wait to apply this to problems I have in other parts of my computer science career.

Shortcomings were a lot. My inability to properly lay filters of an MLP was my biggest issue. I did lose track of time designing and implementing a CNN, and this hurt me on this side. I do understand what happens, and even implemented it on more basic datasets in online compilers. The other biggest issue was not fully understanding what all of the classifications I could go with were. I felt I chose correctly, but it still felt like I was missing data. Even with all of the research, it feels like conceptually I could be better equipped, but perhaps this is in my head. Because of my lack of experience in python as well, It was hard for me to get all of the graphs and data points I wanted, I had to hard configure certain graphs based on the data from my code, instead of having python do it.

What I would like to do in the future is hard to answer. I would like to have been able to more confidently discuss what I did without feeling like I was forgetting to check a box. Typically when I write a report it feels like the hard part is getting all of the data into a small enough quantity to read, I act as a compressor. In this case, I was struggling to fully explain what was in my head to fill the page. This also led me to unpacking the ideas of machine learning and art. If I had more time I would have liked to discuss this a bit in my paper and show some parallels between image classification and art creation.

Finally, I want to address anything in my paper that is not quite what was being asked. I interpreted the project in this way, and tried to uphold what was said in class and on jupyter. My pages match the requirements, my experiments and results are a bit short, but I make up for it in an extended description section. I felt stronger in describing, rather than showing because of my lack of data visualization skills when discussing this project. I had a lot of fun with this project, and honestly expected to get bored, but plenty of problems and exploration were had!

1. Figure 1 on page 12 [↑](#footnote-ref-0)