

Gun Block: Detecting Guns in a Video Stream

Using Machine-Learning

New Mexico

SuperComputing Challenge

April 8th, 2020

Team 13

Capital High School

Team Members:

Edwardo Pena edwardopena@icloud.com (505) 577-0799

German Rojo germanrojo2314@gmail.com (505) 577-7146

Oscar Sandoval Torres oscart0510@gmail.com (505) 316-9406

Teachers:

Barbara Teterycz bteterycz@sfps.k12.nm.us

Irina Cislaru icislaru@sfps.k12.nm.us

Mentor:

David Gunter dogunter@gmail.com

Table of Contents

Abstract/Executive Summary.....	3
Methodology/Research.....	3
Multi-Layer Perceptrons (MLP's).....	4-7
Convolutional Neural Networks (CNN).....	7-11
Mask R-CNN.....	12
VGG Image Annotation (VIA).....	11-12
Results.....	13-18
Conclusion.....	18
Future Work.....	19
Acknowledgments.....	20
References.....	21-22

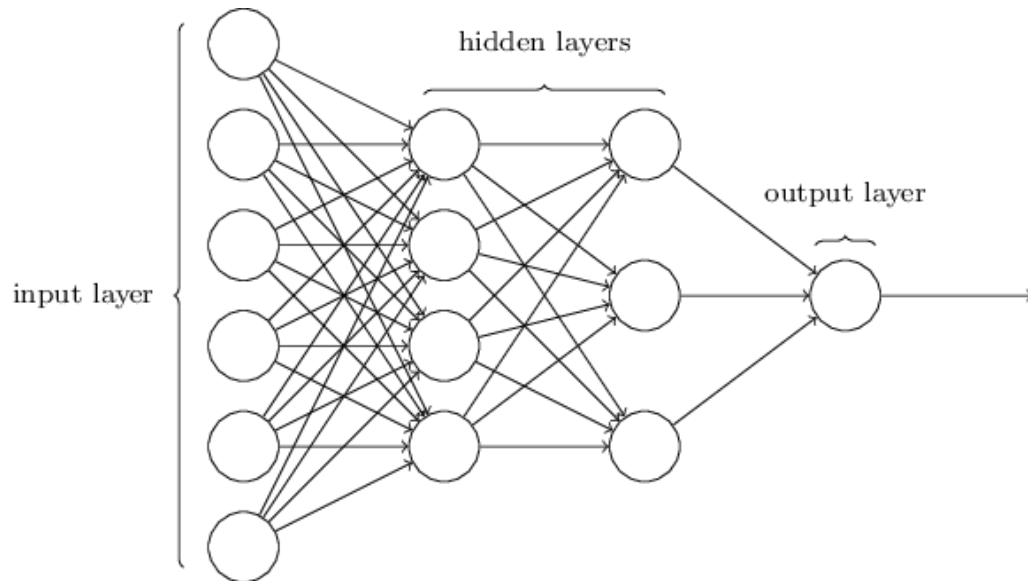
Abstract/Executive Summary

In the society we live in, we find an abundance of taboo topics. Topics that often, while important, are seemingly impossible to talk about, and any attempt often ends in political disagreements. A topic that falls under this taboo is guns. The topic of guns is one that has sprouted much debate and unfortunately little results. The truth is, our political system is slow, which can be bad and good, but at a national level, issues do not seem to get better. This pending frustration became the motivation for this project. We wanted to find a way that we could tackle the problem ourselves. We decided to try to make a gun detection system. With guidance from our mentor, this led us into the field of machine learning and neural networks.

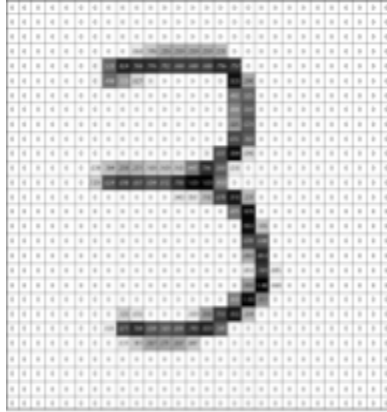
Methodology/Research

The first step in achieving our goal was to get familiar with the topic. Instructed by our mentor, we started off with simple models and gradually moved to something that we wanted to use. The first neural network that we implemented was an MLP (Multi layer perceptron). This network can be divided into three distinct layers: the input layer, the hidden layer(s), and the output layer. Each one of these layers is composed of perceptrons, or artificial neurons. Each perceptron can be simply described as a container holding a numerical value. In order to explain how the MLP network works, let's analyze the example that we implemented to recognize handwritten numbers from one to ten. This example is common and convenient because of the easy access to a database of handwritten digits, called MNIST. This set of data includes around 60,000 images and samples of such digits and is available online.

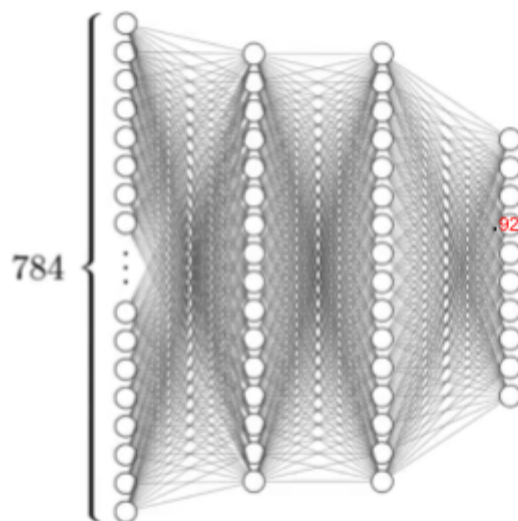
MLPs



The input layer, as the name might suggest, is the layer in which we feed something into the system. This particular something is a value assigned to each pixel of an image based on color. For our data, since we are only considering black and white images, we use their Grayscale number where traditionally black is represented by 0 and white is represented as 1, with grays being a number in between zero and one. But, since our images have much more white than black, in order to minimize the number of operations, we flipped the values (thus making white 0 and black 1). This number is referred to the perceptron's activation, with zero being the lowest and one being the highest. Also, this dataset works with 28x28 pixel images, so our total number of pixel values is 784. This happens so that we take each individual pixel value and reorganize it into a line that then is fed into the input layer.



Before going into the hidden layers, let's first talk about the output layer (as the hidden layers get straight into the math). The output layer is simply a set of possible outcomes in the network. For our example, the output layer would consist of ten possible outcomes: numbers zero through nine. At the end of the process we expect the network to give us a percentage of how much or how little it believes the picture is present in each of the possible outputs. The number with the highest percentage would therefore be the guess. For example, it could look like:



The hidden layers are where "learning" happens. Before getting into all the math, let's talk about what the hidden layer should do. Ideally, we would like the hidden layers to be able to pick out features in the images, like vertical edges, horizontal edges, or loops. In order to calculate the values of a perceptron in the first hidden layer, we take all the activation numbers from the input layer and multiply them by the weight that connects them with the perceptrons in the hidden layer. The weight is a random number, so in order to keep our answer between 0 and 1 we use an activation function like the Sigmoid or ReLu functions. Even though these functions go beyond the high school curriculum, and are usually taught in the college-level, linear algebra, we were able to learn how the Sigmoid function is constructed and how it works. Here is what we've learned: the Sigmoid function ($S(x)$) is equal to $1 / (1 + e^{-x})$, where e is a mathematical constant equal to 2.71828... (it's the limit of $(1 + 1/n)^n$ as n approaches infinity, so that the larger the n in $(1 + 1/n)^n$, the closer to e). Sigmoid function ($S(x)$) squishes the value of x into the range between 0 and 1, which means that the negative inputs into this function produce a result that is close to 0 and positive inputs into this function produce a result close to 1, for example:

$$\text{for } x = -1, S(x) = 1 / (1 + 2.71828...^1) = 1 / 3.71828... = 0.26894...$$

$$\text{for } x = 0, S(x) = 1 / (1 + 2.71828...^0) = 1 / (1 + 1) = 0.5$$

$$\text{for } x = 1, S(x) = 1 / (1 + 2.71828...^{-1}) = 1 / (1 + 0.3678794...) = 0.731...$$

$$\text{for } x = 3, S(x) = 1 / (1 + 2.71828...^{-3}) = 1 / (1 + 0.0497871688...) = 0.95257... \text{ and so on.}$$

Only the highest positive result will light up the corresponding neuron (or node). Knowing this, we were able to understand the following Python function used to calculate the result of the Sigmoid function:

```
import numpy as np
```

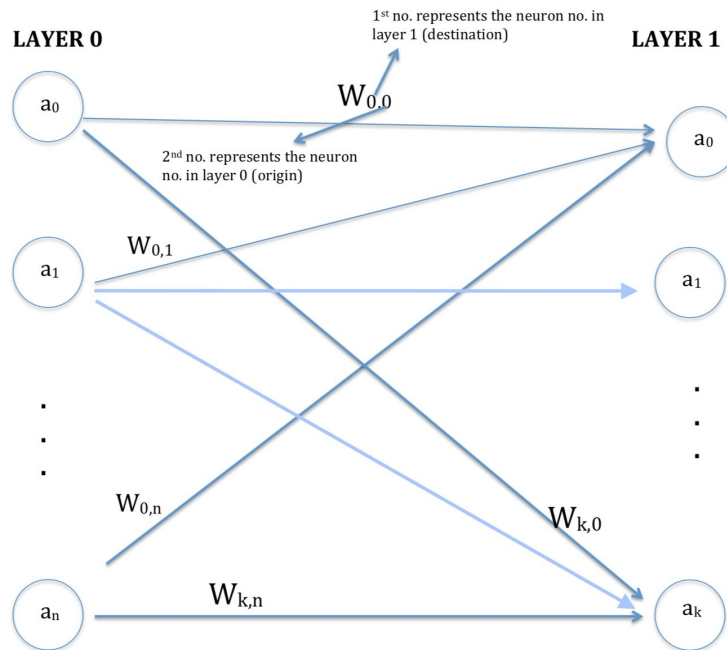
```
def f(x):
```

```
    return 1.0 / ( 1.0 + np.exp(-x) )
```

where: numpy (np) is the name of the Python library widely used by the scientists, f represents the Sigmoid function (with x being its input), and exp(-x), a predefined function from the numpy library, is used to calculate the value of e to the power of what is inside the parentheses, so -x.

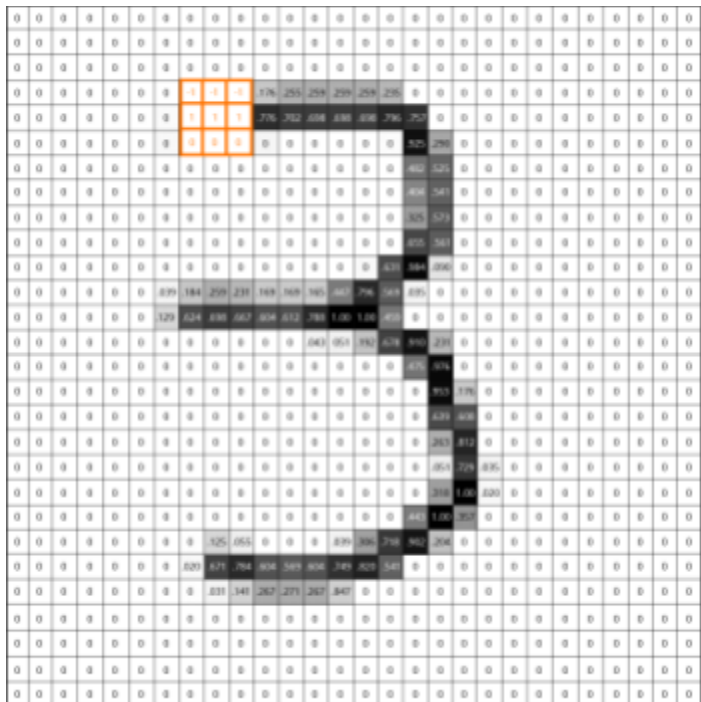
Before using the Sigmoid function we add a bias to the perceptron so that we can set a required amount of activity before it activates. Thus, the equation would more or less look like:

$a^{(1)} = \sigma(w_1 a_1^{(0)} + w_2 a_2^{(0)} + w_3 a_3^{(0)} + \dots + w_n a_n^{(0)} + b)$; where: $a^{(1)}$ is the activation of the first hidden layer, σ the Sigmoid function, b is a bias, and w_n is the weight that connects the neurons in layer 1 ($a_n^{(1)}$) with neurons in layer 0 ($a_n^{(0)}$), as represented by the image below.



The image above displays an example of a filter and where the filter would start.

Calculating the dot product for the scenario illustrated is straightforward and you will end up with zero. This zero is then taken and placed on a new pixel map. After this is calculated then we move the filter over depending on the chosen stride length. For the sake of simplicity, let's do a stride length of one but skip some time into the process where things get more interesting.



-1	-1	-1
1	1	1
0	0	0

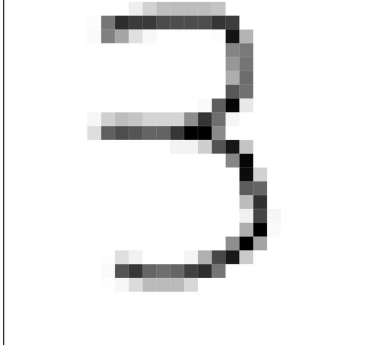
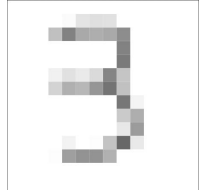

•

0	0	.063
.533	.824	.769
.498	.125	.027

$$= (-1 \times 0) + (-1 \times 0) + (-1 \times .063) + (1 \times .533) + (1 \times .824) + (1 \times .769) + (0 \times .498) + (0 \times .125) + (0 \times .027) = 2.054$$

Things get a bit more practical when we reach a place with numbers other than zero. We do the exact same thing and calculate the dot product. The process continues until you have a new chart. This might not be obvious at first but due to the process not being able to do dot products of the outer layers, the final chart ends up being 26x26 pixels. In order to fix this we do what is called padding; which basically just adds a layer of zeros around the new pixel map.

feed our training data into the system. This process can be tedious and long, so it is important to maximize efficiency where we can. One of the ways this is done is to do pooling before passing the filtered images to the next step. In essence, this process shrinks the image without compromising the details too much. We choose a pool size and then, in the similar way to the convolutional layers, we slide a filter across the image to reduce its size. For example, with a pool size of 2x2 you would take the average value, for regular pooling, or the maximum value, for max pooling, of a 2x2 section on the upper left of the image and place the new value in a new image. Typically, the pool size is also the stride length.

		
Original 28×28 image	Average-pooled 14×14	Max-pooled 14×14

Since our images are so small, the max pooling has the effect of removing too much detail and even regular pooling is cutting it close. “In the MLPs, the process was to multiply input nodes by weights, add in a bias value, then apply the activation function. In a CNN, the weights are the filters that form the convolutional layer. They are applied during the convolutional step. Bias

values can then be added to the results and activation functions applied. That is, each element of the resulting convolved image data has a bias value added, and then each value is run through the activation function, producing another set of data, activated data.” (Gunter, D. 2019).

Convolutional Neural Networks). This activated data is then squished like you saw in the first step of the MLP system and follows that structure afterwards. The difference is that now every perceptron in the hidden layer corresponds directly to the presence or absence of some feature in the image.

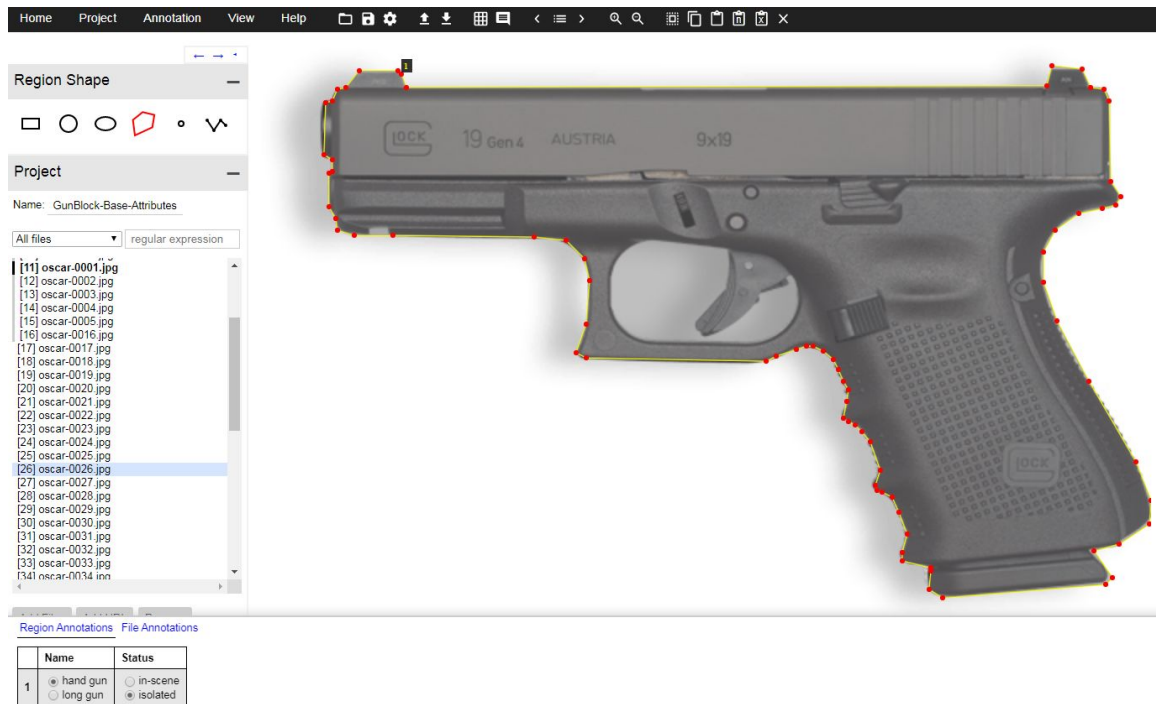
Mask R-CNN

For our final model we ended up using a Masked R-CNN. The “R” in the name stands for the region. This CNN adds a feature where it tries to predict objects of interest in the image. The Mask part of the CNN basically tries to classify every pixel to see if it is a part of an object. It does so by comparing color and texture among other things. In essence, it takes pixel mats that have been created by convolving and/or pooling the original image and it maps them as a mask on the original image by a process called ROIAlign. The other reason we chose it is that it has very impactful results with the masks making for some very neat visuals. To be perfectly honest, there is a lot about Masked R-CNNs that we are yet to fully dive into and understand.

VGG Image Annotator

VGG Image Annotator (VIA) is a simple manual annotation software that can be used for images, videos, and audio. VIA is an open source annotator that is based on HTML that runs in a web browser and does not require any installation. In this case, VIA annotation was used to annotate our gun images. First, we needed to upload our images to the VIA annotator. Next, we

created attributes by indicating if the images displayed a handgun or a long gun and if they were in-scene or isolated. We then used the region shape tool to trace around the gun by marking points as shown in the image below.



We needed to do this for all guns that were in the images. After annotating hundreds of images the annotations were saved as .JSON files, which we then used as inputs to our machine learning (ML) system.

Results

As we fed images to our ML system we received various types of results. Under each image we have provided a description of our results as well as why we believe our results came back the way they did.



In this case, the gun was detected almost perfectly. Our system was able to outline the gun properly due to all the images we used to train our system and the various angles of the images we trained it with.



In this image, our system was confused due to the shape of the wooden clip. The shape of it resembles the barrel of a gun, and the bottom part is similar to the handle of a gun.



In this image our system did not detect anything. We believe this was because we did not feed enough images of the front of a handgun into our system, which is why it did not recognize anything.



This was another great result since our system was able to properly detect the gun. We used many images showing a hand gun from the side in our training set, which is why our system was very accurate in this particular scene.



In this image our system was able to detect the handle of the gun properly but it missed the rest of the barrel of the hand gun. We believe that this is due to the contrast in the image. Since the background is also black, it had trouble detecting the rest of the gun.

Predictions



This system found the gun fairly well and outlined it well despite the complexity of the gun. The only issue was that it thought it was a handgun and not a long-gun. This was because most of our current training set has been hand guns with a limited amount of long-guns.



We also used some images that were definitely not guns and our system did a great job getting it correct.

Conclusion

This has been an incredible process and opportunity for us to attempt to better the world we live in. We learned how to approach a massive problem through hard work, trial and error. We learned about various types of neural networks and through research and trials we found that Mask R-CNNs were a great way to approach this problem. Through this long process we came away with a well functioning system that can detect guns very well, especially handguns. We learned the values of teamwork and research to help guide us through the process. Although our system is not perfect, we have worked hard to create a system that will serve as a step to a better future.

Future Work

As we mentioned before, our system is not perfect. This is a massive project that could take years to improve as we continue to do research on machine learning and image recognition. Something that we can do in the near future is to feed more images from various angles and different backgrounds into the system. This would improve the accuracy as well as give us better results all around. Our system struggled with identifying guns from the front, which should be fixed as we feed more images of guns from the front into the system. In the near future, we could also train the system with more images that have difficult backgrounds so that it can find the gun in complex scenes. Overall, we hope to improve the accuracy of the system without compromising its speed. Our ideal system would be able to detect a gun fast enough in a live video stream, so it will help protect people and make the world safer.

Acknowledgments

This project required hard work and determination from our teachers and mentor. We would like to thank Ms. Cislaru for keeping us updated on deadlines and her encouragement along the way. We would also like to thank Ms. Teterycz for keeping us on track and pushing us to do better. Lastly, we would like to thank Dr. Gunter for his kindness and willingness to mentor us through this project. Dr. Gunter has provided us with many resources and his incredible knowledge. He has encouraged us to do our best throughout the project. Our project was a fascinating opportunity which would not have been possible without our amazing teachers and mentor. Thank you!

References

Gunter, D. (2019). Intro To Numpy. Retrieved from

<https://colab.research.google.com/drive/1grcZ2LPnAxfl73oPTiDo6efW0tg221AB?authuser=1>

Gunter, D. (2019). Neural Networks Using Python . Retrieved 2019, from

<https://colab.research.google.com/drive/1H8189uj2236b1i8IwnMf9ZnXfM4xmEpA?authuser=1>

Gunter, D. (2019). Convolutional Neural Networks. Retrieved from

<https://docs.google.com/document/d/1sxic18ZKLflhiusLrs0zUkxH-xtjxAa3zs70aQz5r0/edit>

Gunter, D. (2019). MNIST_tensorflow. Retrieved from

https://github.com/dogunter/MNIST_tensorflow#mnist-tensorflow

Gkioxari, Piotr, Girshick, & Ross. (2018, January 24). Mask R-CNN. Retrieved from

<https://arxiv.org/abs/1703.06870>

Gunter, D. (September, 2019). Introduction to Artificial Neural Networks: Part 1. Retrieved from

https://docs.google.com/presentation/d/1U8KGCWQ9ciAiqp5zuTc8BaAKB7iToKfGtrSHQIp03wE/edit#slide=id.g5fbf8eefbb_0_10

Lai & Maples. Developing a Real-Time Gun Detection Classifier. Stanford University.

Retrieved from

<https://drive.google.com/open?id=17OCUfTmImOS0H-PP5bUrSmGJJp6nN-a7&usp=gmail>

Matterport. (2019, March 9). matterport/Mask_RCNN. Retrieved from

https://github.com/matterport/Mask_RCNN

Nielsen, & A., M. (December, 2019). Neural Networks and Deep Learning. Retrieved from

<http://neuralnetworksanddeeplearning.com/>

Zhang, X. (2018, April 22). Simple Understanding of Mask RCNN. Retrieved from

<https://medium.com/@alittlepain833/simple-understanding-of-mask-rcnn-134b5b330e95>