Digit Detection Modeling Basic Search and Recognition Mechanisms

Prabhdeep Samra

COGS 123, COGS 223, CSE 173, EECS 273 :Computational Cognitive Neuroscience

Dr. David Noelle

May 8, 2018

Abstract

The goal of this project is to produce a network that can accurately locate and identify all digits on a given input image. The network is a simplified model of the human ventral perceptual system by only modeling the V4 and IT-posterior. This means that the network cannot learn at a significant level. Instead, it simply models how people recognize levels from a high level of abstraction. The network looks at small portions of a given input image, called “windows”, and has 10 receiving units that respond to the current window. Each receiving unit corresponds to a digit (0-9), and an activated unit represents that a respective digit is currently present in the window. The project resulted in a simple, slow-working model for digit detection with basic search and recognition mechanisms using ten detector units that respond to a series of given inputs, representing how neurons in the visual system respond to seeing and recognizing numbers. The network was successful in detecting digits and filtering out other objects such as shapes, but had trouble dealing with noise, which caused false positives in receiving unit activations.

Introduction

The motivation of this research comes from a diversion towards digital images and computer vision. Essentially, the goal of this project is to create a primitive optical character recognition (OCR) for numerical digits.

Digit recognition for the average human being is a seemingly trivial task, yet in some cases it requires the use of many regions in the brain. In the human brain, the ventral perceptual system is split up into 5 regions: V1, where edge detection occurs, V2, where these edges are combined to form basic shapes, V4, where the shapes are translated into features, IT-posterior, where the shapes along with their features are encoded into objects such as people, and IT-anterior, where the objects become associated with their semantic meaning [1]. The objective of this research is to model this phenomenon using ideas from computational cognitive neuroscience. The task at hand is to find and recognize any numbers present on a given image.

A rudimentary approach to modeling this phenomenon involves viewing the problem from a high level of abstraction. For any given digit (0-9), it either is or is not said digit, e.g., the number “4” either is, or is not “4”. Though some digits can be viewed with a strong likelihood with each other, and often even humans have trouble distinguishing among them (e.g., a poorly drawn “4” and “9”), this basic idea is the backbone of the digit-detecting network. This methodology does not model every aspect of the human perceptual system. Instead, it simplifies the model, including only V4 and the IT-posterior, meaning the looks at and identifies objects as their wholes, without breaking them down into their baser features. The image being shown represents what is in the V4, and this is compared to a small library of objects contained in the IT-posterior.

To find digits in a given image, there must be some background for the network to work with. In this scenario, images are black and white. The white pixels on the image represents the background, or nothing. The black pixels on the image represents something, or part of something. This just means that digits and other objects will be drawn in black on white background. Images are made up of a single pixel matrix (2D) and can be thought of as a grid. Each pixel has an x and a y location on the image and can be manipulated or used as data.

The Methods section of this paper will discuss the nature of the network and describes in detail how digits are picked out from a given image, and how digits are differentiated from one another. In the results section, the output of the network will be revealed, which will be evaluated in the discussion portion, along with some concluding remarks.

For a detailed explanation of how to run this network, refer to the appendices, which contains detailed instructions on how to download the source code and other relevant files, along with directions on how to generate data sets and how to start the network.

Methods

The Network Model

The structure of the network model used is based on Randall C. O'Reilly's "Detector" simulator [2]. The network starts with a given image and an input. The input is simply a small portion of the image, which will be referred to as the "window". The input to the network is connected to ten receiving units, one for each digit. This is depicted in the figure below.

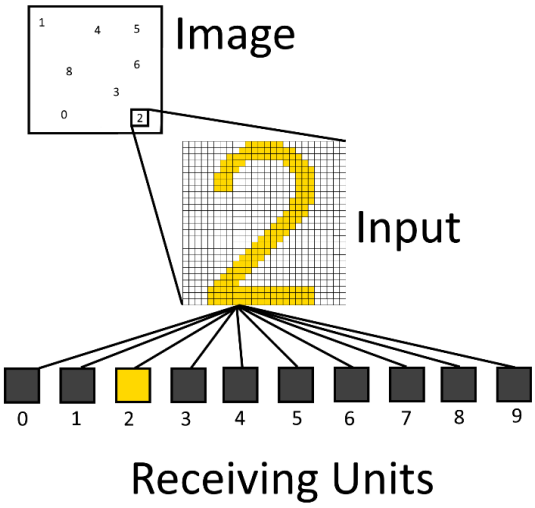

Figure 1: Model representation of the detecting portion of the network

The given image is 700 x 700 pixels wide, and the window is 26 x 26 pixels wide. Only one window can be used as an input to the network at a time. To detect digits on the entire image, the window "scans" through the image, starting at the top left corner of the image, and moving over one pixel at a time. This scanning process is sometimes used by people when involved in an activity that requires visually searching, such as a word-search. This is illustrated by the figure below.

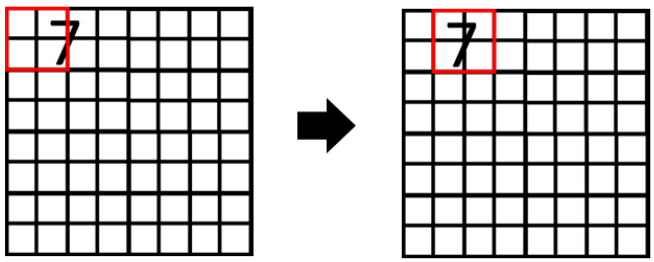

Figure 2:Representation of how the window moves across the entire image. On the left, the window only selects part of the digit, on the right, the entire digit is encapsulated by the window

Each time the network gets a new input from scanning window, the receiving units respond. The receiving units uses a 26x26 pixel wide template (the same size as the window) to determine whether or not to activate. Each receiving unit activates based on the sum of differences between the template used for the receiving unit and the input. This difference decides the threshold value of the receiving unit, which will be discussed in further detail later. Ultimately, if the difference is small enough, the receiving unit actives which signifies the respective digit was detected.

Figure 3: Templates the receiving units use to determine if a digit is present. Notice how "grainy" they look. This is because there are no grey pixels to smoothen the image out, and because the image is low in resolution.

Preprocessing

The structure above alone is not a satisfactory model of how humans visually search for something. No matter how many digits appear on an image, the network will take the same amount of time to process (a rather long time), which does not accurately model the human visual system. To decrease the amount of time the network takes to detect digits, a preprocessing masking step is implemented. This preprocessing step is inspired from an observation in the human visual system. In an image such as in the left one in figure below, a person need not scan the entire image to recognize that there is a triangle in the center of the image. Humans have the ability to quickly focus attention to regions of interest, which inspires the idea of region masking.

The method of region masking is simple. A square grid is set upon the given image on the left, resulting in the middle image. From this regionalized image, the network searches pixel by pixel to see if there is anything in each region (keep in mind that black means there is something on the image). If there is anything in the region, no matter how small, the entire region is marked to be scanned. This is depicted by the figure on the right, wherein each region the triangle lies in is highlighted. This preprocessing step models the dorsal split in visual processing, as it involves spatial location information.

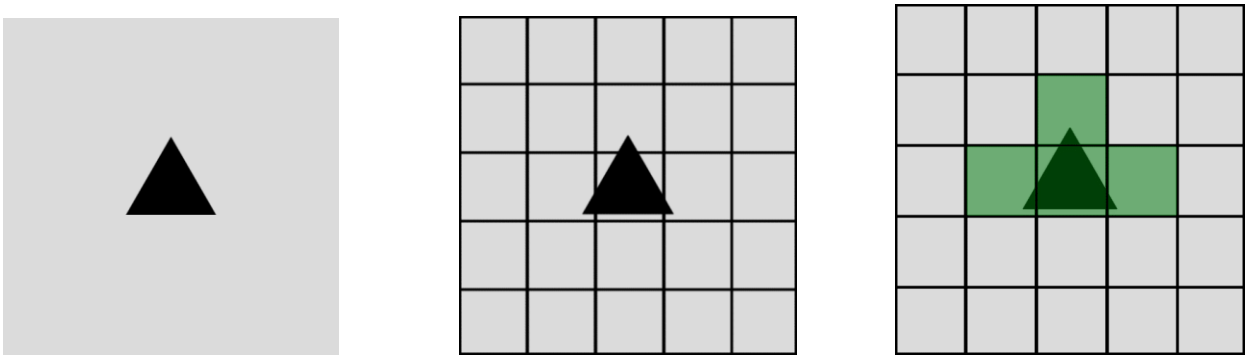

Figure 4: Visualization of the masking process. The input image on the left has a grid transposed above it (middle), then, the network marks which regions have parts of the triangle in them

Learning Algorithm

This network is not a traditional network in way that it does not have any layers other than the input and receiving units, so there is no real capacity to learn. There is however, an adaptive method of setting the threshold value for activating the receiving units. The receiving units could be set only to fire at 100% similarity between the current window and the template the receiving unit is set to fire at. This is because the image (and all the windows therein) are made using the templates in the receiving units. However, this means that no receiving units will activate if there are variations in the digits on the images. A dynamic method of setting the threshold value involves the network learning from a data set of 10 images. Each of these images are 26x26 pixels wide, and each has the digits 1-9 on them. These digits should be "damaged" in some way, as to simulate how they might look on a real image. These images can be thought of as the most different a digit can be, while still being detected. From this, the network can look at the difference between each image in the data set and the receiving unit templates. The difference can be mapped to similarity by using the following equations:

$$D_{Nmax} = Max(\overrightarrow{D_{NM}}) \qquad \text{(where M} \neq \text{N)}$$

Where $\overrightarrow{D_{NM}}$ are the difference between a single receiving unit template and a digit from the data set. In this process, each value in $\overrightarrow{D_{NM}}$ is calculated by summing the absolute value of the difference between every pixel in the receiving unit template and digit template. For example:

$$D_{N0} = \sum_{x=1}^{26}\sum_{y=1}^{26}\left|R_{0xy} - P_{0xy}\right|$$

Where R_{0xy} is the pixel value from the "0" receiving unit template at location (x,y), and P_{0xy} is the pixel value from the "0" data set template at location (x,y). From this value, we can calculate the similarity of a given window and a receiving unit template with the equation:

$$S = -\frac{100}{D_{Nmax}}x + 100$$

Where x is the difference between a non-data set window and a receiving unit template. Using this equation, the threshold value can be set by using a percentile system. For example, if the network was only to detect digits that were at least 90% similar, S would need to be greater than or less than 90 for a digit to register. This process can also be used for dynamic thresholding. The network will be fed a set of training templates which should be "damaged" or "obstructed" in some way, and it will determine the best thresholding value based on them. In simple terms, the threshold value should be between the response rate of the receiving unit in question, and the next highest response rate. The thresholding process is depicted in figure 5 below. A lenient threshold value for the digit 0 is the highest similarity rate (response value from the receiving unit) from the receiving units that is not 0 itself. Ideally, the threshold value is somewhere between the response rate from the receiving unit corresponding to the digit

and the next highest response rate from the other receiving units, as to only activate one receiving unit. Using this method, each receiving unit can have a different threshold value.

Figure 5:Receiving unit response rates for the digit 0. Notice how the strict threshold value is the response rate from the receiving unit corresponding to the digit being detected. The lenient threshold value, in this case, is 4, which is the most similar digit to 0.

Data Set

The data sets the network runs on are images with digits/shapes/noise that were procedurally generated using the templates in figure 6 below, along with a global noise function that adds black and white pixels randomly across the entire image.

Figure 6: Templates used to generate images

To create the data set, an algorithm runs through the entire image, and at random intervals, decides to draw a random digit/shape. There is a mechanism in place to make sure that the randomly placed digits and shapes do not overlap with each other, both to simplify the data set, and to make it easier on the detection network. The rate at which templates are drawn onto the image can be controlled by adjusting the range of uniformly distributed random integers used to determine whether or not to draw. Examples of these procedurally generated images will be shown in the results section.

The templates used for dynamic thresholding are shown below. These were created by adding salt-and-pepper noise to the templates in figure 5 above.

Figure 7: Templates used for dynamic thresholding. These templates are compared to those in figure 6

Results

A series of four trials were conducted on the digit recognition network, each consisting of a different data set or preprocessing step. The first data set contained images generated only with the templates used by the receiving units, and was tested without using the masking step. The second data set also only contained images generated only with the same templates from the receiving units, but used the region-making preprocessing step. The third data set consisted of images with both digits and basic shapes. The fourth data set consisted of images with digits and shapes, with the addition of noise. Each data set contains ten images fitting the description above.

Running the first data set proved to be successful. As seen in the figure below, each digit on the input image on the left was redrawn on the image on the right. It did, however, take 18 minutes for the network to finish processing just one image. The network was able to locate and recognize each digit from the data set with a 100% success rate.

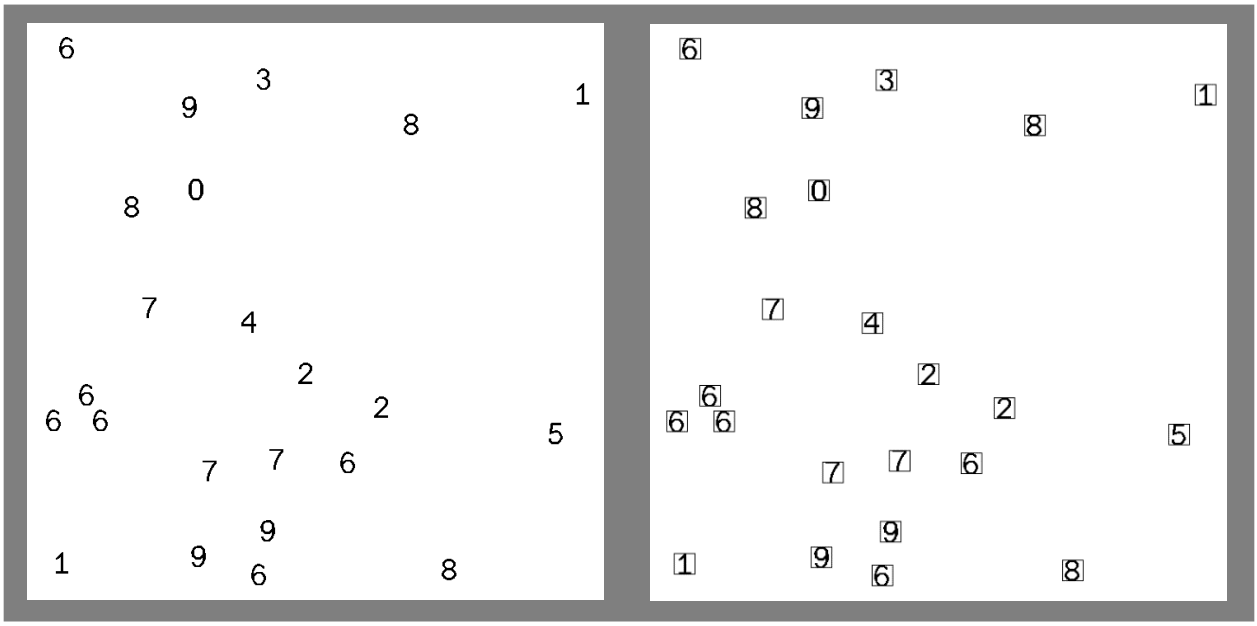

Figure 8: Example results from the first trial

The image on the left was the image fed to the network. The image on the right is what the network outputted when it was finished running. It is programmed to draw the number it "thinks" it recognized inside of a box. The results from the entire data set are shown in figure 8 on the right.

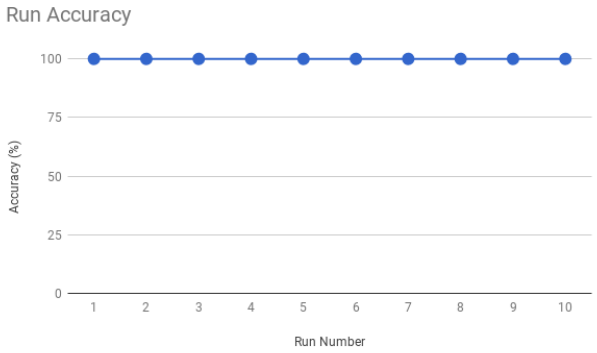

Figure 9: Accuracy Results from the first trial

The second trial also proved to be successful. The only difference between this trial and the previous was the addition of a region-masking preprocessing step. Just as in the last run, the network was able to recognize and locate each digit from the data set with a 100% success rate. However, it did so at a much faster pace. For this particular image, it took the network only 4 minutes to complete. As seen in figure 10 below, the masking step occurs before the detection. The network only scans the regions in black and skips over the white parts. The amount of time this saves is proportional to the number of digits present on the image.

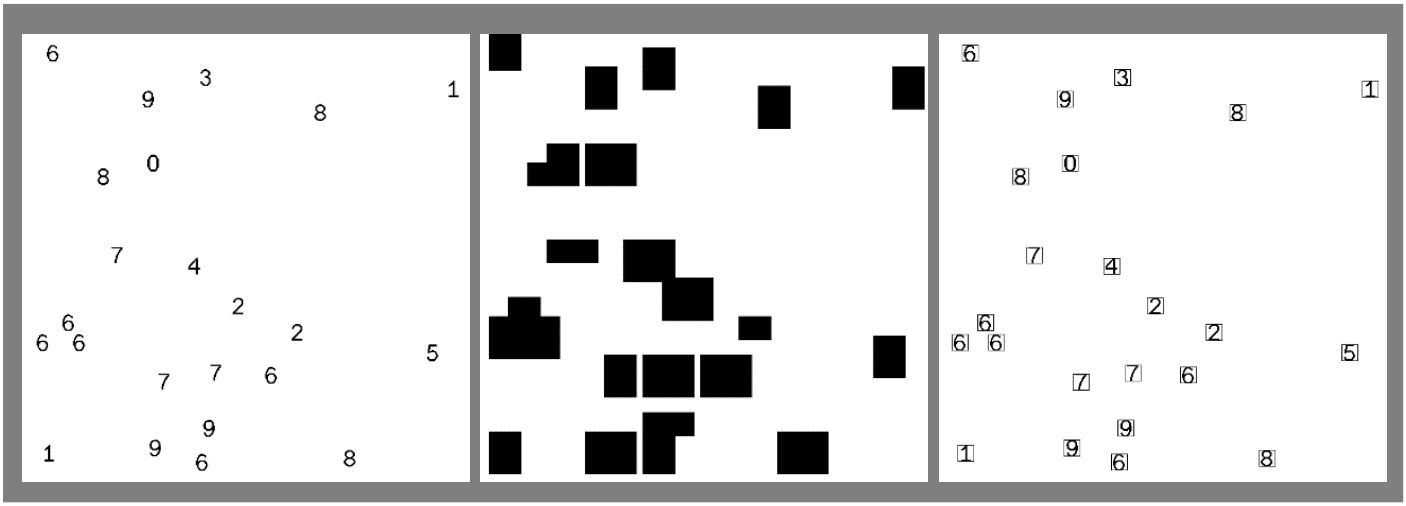

Figure 10: Example results from the second trial

This run was essentially the same as the first, except it was much faster, which was the desired goal of the masking process. As seen in figure 11 below, each the network was able to find and recognize each digit, just as in the first trial.

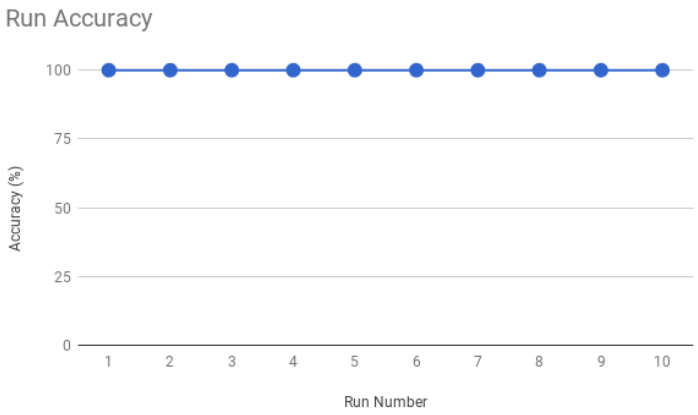

Figure 11: Accuracy Results from the second trial

The third trial was successful. The only difference between this trial and the second trial was the addition of two shapes- solid triangles and hollow squares. Just as in the last run, the network was able to recognize and locate each digit from the data set with a 100% success rate, while the shapes are skipped over. This is the ideal response since the network is not supposed to pick up on anything except digits. Notice how the region-masking step picks up on the shapes as well as the digits. This is because the masking step is not able to differentiate between shape and digits. It simply suggests that there is something in the region that the network should look at. This trial also utilized masking and took 6.5 minutes for the network to completely process the image below. There are many more digits and shapes on this image than on the figure from the second trial, which accounts for why it took 2 more minutes to complete.

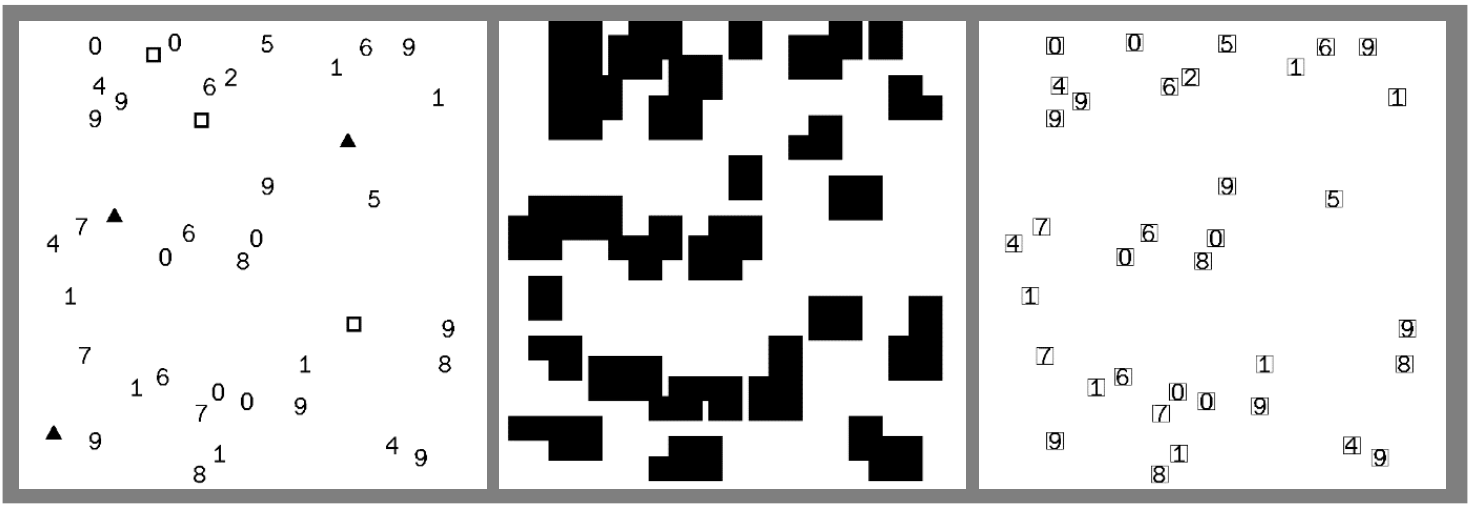

Figure 12: Example results from the third trial

This run was akin to the second, except the input images included shapes. As seen in figure 13 below, each the network was able to find and recognize each digit, just as in the previous trials, while also ignoring the shapes.

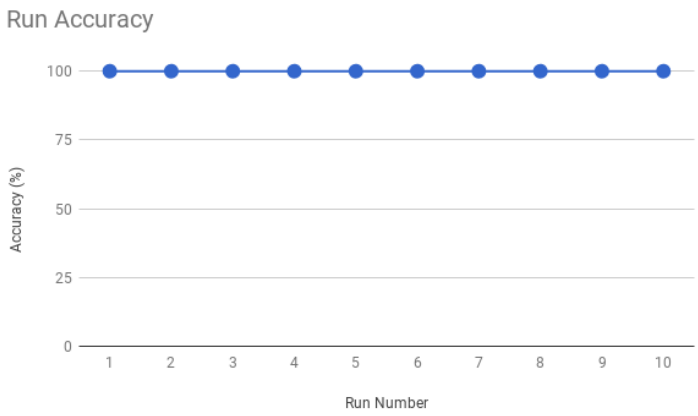

Figure 13: Accuracy Results from the third trial

The fourth trial is where things become more interesting. This trial included images with both shapes and noise added, as seen on the left in the figure below. The noise forced the region-masking preprocessing step to mark nearly the entire image to be scanned, as seen in the middle image. Once processed however, the network was able to detect every digit that was in the original image, though it sometimes got the position wrong by 1 pixel, e.g. it thought the number was shifted to the left/right by one pixel unit. The network did, however, fail to ignore the shapes that were added in. It consistently recognized the solid black triangle as the number “4”, and the hollow square as the number “1”. This is an expected result, given how the shapes can have similar characteristics to the numbers. If the dynamic thresholding was adjusted to be stricter, perhaps the network would follow its intended result and ignore the shapes

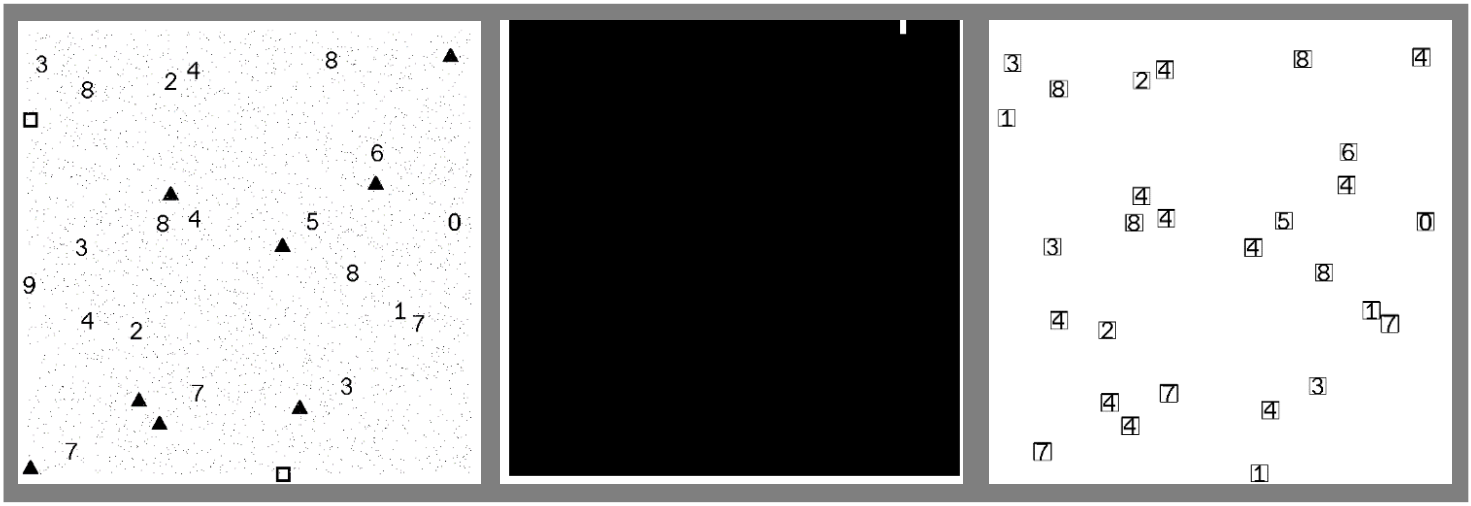

Figure 14: Example results from the fourth trial. Notice how the masking stage marked nearly the entire image for detection

The accuracy rates for this trial are shown in figure 15 below. Detecting a shape deducted points from the accuracy rating. Since the network was still able to detect every digit, the accuracy is high for images with few shapes. The runs with low accuracy are from images that contained many shapes relative to how many digits were present.

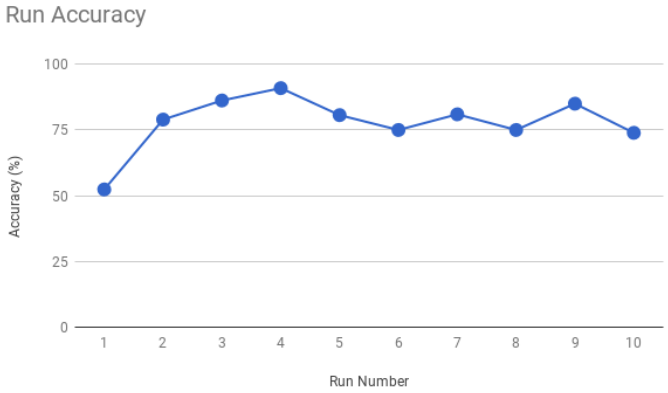

Figure 15: Accuracy Results from the fourth trial

Discussion

Network Success rate in relation to templates/shapes/noise

As mentioned earlier, the network often had a 100% success rate because the templates used to create the images are the same as the templates used by the receiving units to calculate similarity. This means that whenever a digit is shown to the receiving units, the receiving unit that corresponds to the digit will activate with 100% of its energy. This is exemplified in figure 16 below.

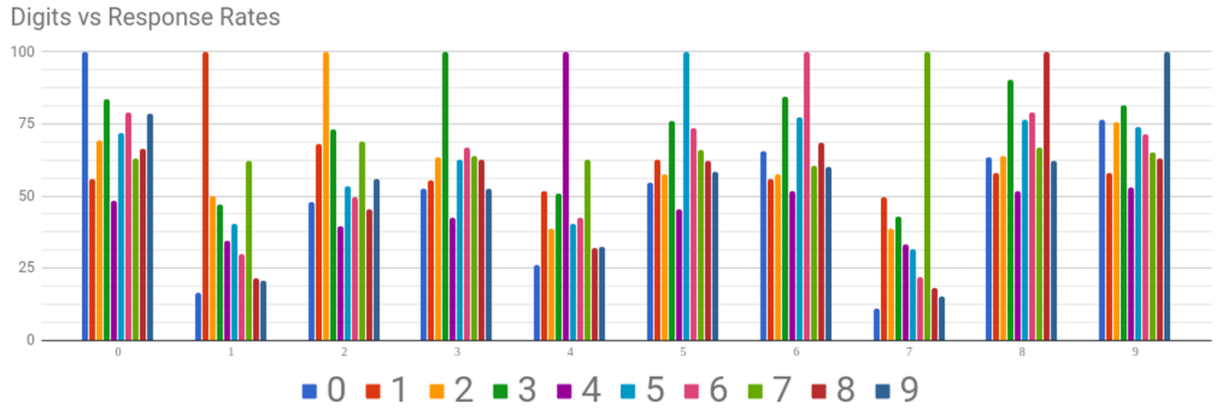

Figure 16: Response (similarity) rates each receiving unit had when shown every digit

Each receiving unit has a 100% similarity rate with its respective digit, which means when the threshold value for a receiving unit is set to 100%, only the correct receiving unit will fire. This is also why the shapes were not detected in the first three trials. In the fourth trial, however, dynamic thresholding was used ("training" the receiving units on "corrupted" digits), because the image had noise added, meaning static thresholding would fail, resulting in several errors by the network. For this trial, thresholding values were set below 100% because of the dynamic thresholding algorithm. This is shown in figure 17 below.

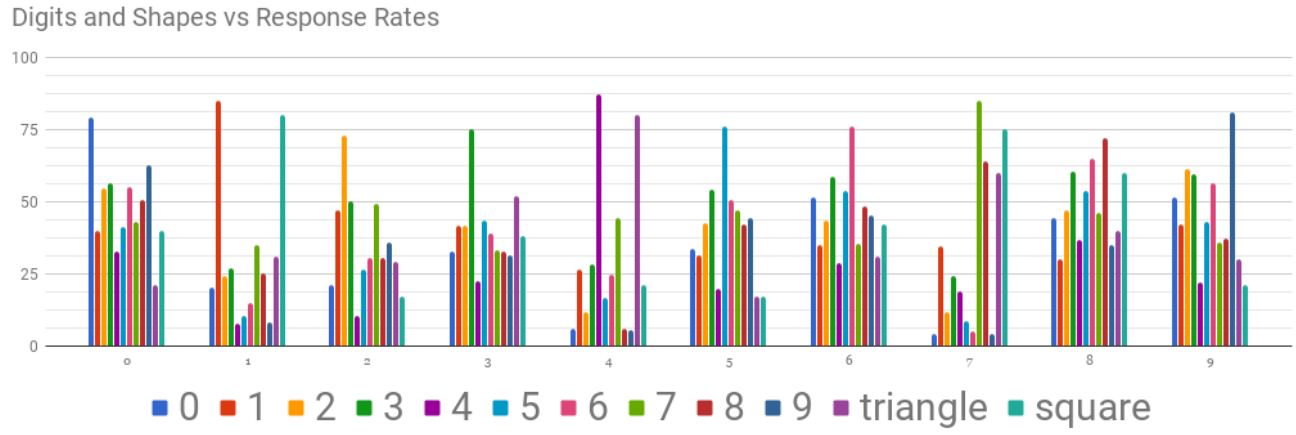

Figure 17: Response rates each receiving unit had when shown every digit as well as shapes given a "corrupted" dataset. Notice how well the triangle responds to the "4" receiving unit, and how "1" responds to the square unit.

In the fourth trial, the network chose thresholding values based on the chart above. When the network was run, it chose values that were too low, which resulted in receiving units responding to objects that were not digits. This effect could be remedied by either forcing the network to use stricter thresholding values, or by using a training dataset that is more representative to the images being shown.

The effect of region masking

As mentioned in the results section above, region masking significantly reduced the time it took the network to finish the task. This is because the masking step allows the network to skip over parts of the image that contain no objects. Therefore, the amount of time saved by the masking process is proportional to the number of objects on the image. When the datasets are generated, the probability of an object being drawn at a pixel location is 1/15000. Multiplying this by the size of the image, 700x700 pixels, yields an average of about 32.6 objects per image. This number is accurately reflected in the actual number of digits and shapes generated in the pictures. Since each object generated is 26x26, and there are ~33 objects on each image, that means the total area occupied by objects is 22308 pixels square. Dividing this number by the area of the image results in the objects taking up about 6.06% percent of an image. However, the masked area takes up much more space, this is because each object can occupy 4 or 5 regions in the masking process. This means the masked area covers about 24-30% of the image, which results in a theoretical speedup of the same percent. In the example used above, the masking process reduced the amount of time the network from 18 to 4 minutes. This is about a 22% speed increase, which fits it with the theoretical value above. This also means that less objects on the image means the speed increase is more substantial, and that more objects on the image means that the masking process did not have as much of an effect, as seen in trial 4 from the results above.

Conclusion

Overall, the network was successful. It demonstrated its capabilities on a variety of data-sets, and although it struggled with images with added noise, adjustments to the thresholding values may improve overall network accuracy. The network's region masking mechanism models the dorsal split in visual processing, as it involves spatial location information. When people look at a large image, their focus is immediately diverted to regions of interest, which is what the masking process achieves. The network's "scanning" mechanism models how people might process vision when involved in an activity that requires visually searching, such as a visual puzzle or exercise. Dr. R.C. O'Reilly wrote about how people scan "using a 'spotlight' of visual attention to focus on small areas of the image, which can then be processed effectively by the object recognition pathway" [2]. This, in effect, is what the network does, it just takes much longer for the network to do.

The project resulted in a simple, slow-working model for digit detection with basic search and recognition mechanisms using ten detector units that respond to a series of given inputs, representing how neurons in the visual system respond to seeing and recognizing digits.

Acknowledgements

Dr. David Noelle for consulting and feedback about the project

Dr. Randall C. O'Reilly for his writings and simulators

References

1. O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors (2012). Computational Cognitive Neuroscience. Wiki Book, 1st Edition. Perception chapter. URL: http://ccnbook.colorado.edu

2. O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors. "Detector." CCNBook/Sims. URL: https://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Sims/Neuron/Detecto

Appendices

Everything needed to replicate this project is as follows:

- MATLAB
- An Internet connection

The Matlab code used for this project, as well as a pre-made dataset and receiving unit templates can be found at:

https://github.com/prabhdeepsamra/Digit-Detection-Network

To generate data sets, follow these steps:

1. Download all the files and extract into a directory
2. In Matlab, navigate to the working directory and all the folders to the selected path
3. Open imageGenerator.m and select run
    a. For options in this program, read the comments in the code
4. From here, a series of 10 images will be generated

To run the network, follow these steps:

1. Steps 1 and 2 from above
2. Open DetectorNetwork.m
3. Set the desired image in the code (see comments)
    a. For options in this program, read the comments in the code
4. From here, the network will detect digits from the image selected in the above step

There are several support files that the DetectorNetwork utilizes. To use region-masking, mask.m and segment.m are used. Segment.m splits up the image into regions and records into a vector the location of each region that contains an object, or part of an object. Mask.m takes these coordinates and writes each of the regions as valid for the DetectorNetwork to run over. To use dynamic thresholding, DetectorNetwork calls threshold.m, which takes in a dataset of 10 images, and the receiving unit templates. It generates the threshold values that the receiving units must exceed to activate.