EXPLORING REPRESENTATION LEARNING THROUGH ALTERNATE LENSES

Benjamin Vincent Cutilli

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Master of Science

in the Department of Computer Science,

Indiana University

December 2021

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Master of Science.

Master's Committee

Professor David Crandall, PhD

Professor Donald Williamson, PhD

Acknowledgments

David Crandall was the advisor of this thesis, and was helpful in many ways (including, but not limited to, what should go into the manuscript, super accommodating when personal issues have arisen, and general advice for thesis). He and Professor Donald Williamson (Indiana University Bloomington) reviewed this thesis, so for that I am grateful. Sven Bambach (a postdoc at the time) talked with me about subitization with networks, which surely helped with the thesis. And, finally, my family, friends, and others who have helped for the general support they have provided during what, personally, was a difficult time for the author. Thank you, everyone.

Benjamin Vincent Cutilli

EXPLORING REPRESENTATION LEARNING THROUGH ALTERNATE LENSES

In this work, we take a look at representation learning. To do so, we perform experiments in shape counting in the hopes that the artificial neural network learns the representation of shapes. The network, at least in part, appears to learn some representation resembling the shape. We formulate a way to perform shape detection using prior-research's saliency maps in a novel way, though the algorithm has issues to be addressed at a later time. Another representation topic that we look at is that of adversarial examples: if a person wants to reduce the classification performance of a neural network substantially or completely, there are ways to turn normal images into "adversarial examples" without a human noticing the change. We propose a number of defenses and both analytically and intuitively state their properties that would, in theory, lead to them performing well. However, the theory did not pan out in any substantial way, but we make some conjectures as to why this is the case. In all, our the experiments are semi-successful and we extensively discuss how these topics rely heavily on representation.

Professor David Crandall, PhD

Professor Donald Williamson, PhD

Contents

Chapter 1

Introduction

Neural networks have been an indispensable tool when it comes to artificial intelligence. They have made possible the automation of many tasks that were previously impossible [Red+16; Rus+15]. Due to the Universal Approximation Theorem [HSW89], neural networks are theoretically able to perform any task, and thus we find them effective at solving previously unsolved problems. Mainstream training algorithms for neural networks, such as Stochastic Gradient Descent, require lots of data for the neural network to learn an effective function. However, the definition of "effective" does not necessary mean that the learned function truly represents the data on which it is trained [Sze+14b]. While some work (for example, [Yos+15]) has shown that this seems to be the case, other evidence [Sze+14b] suggests the exact opposite. Clearly, more work into which training for proper representations is examined is necessary.

In this thesis, we explore some interesting points regarding representation. The first subject we discuss is *subitization* [Kau+49; Ner14], a counting method that humans possess in which the process of keeping track of an internal counter of each instance of an object (or "each object instance"; the difference is discussed/debated later) is short-circuited, the count being directly determined by what the view containing the object looks like. The question we pose is: can we train a neural network to do the same, and does the learning process translate into the network actually understanding what the object looks like? Further, how can we even determine that a network has such properties in the first place? If a proper representation exists, can we localize each instance of an object?

In the second part, we directly address the aforementioned issue brought to light by [Sze+14b]. We take a look at multiple types of defenses, one of which modifies the loss function, a commonly used strategy [GSS15; KKG18; Mad+19], and three structural techniques, one of which modifies

the former non-structural defense. While not successful, at the very least, we analytically asses the defenses to provide some kind of justification for them that structural changes can produce some sort of defense against adversaries. As a result, it may make sense for future research to consider different types of networks as an alternative.

These two subjects probably do not do much more than scratch the surface of the topic of representation learning, but hopefully it injects some amount of motivation into the field to further explore the topic. The author believes that not enough focus has been on the core concepts that underpin representation, which is backed up by the general consensus that neural networks are a mathematical black box.

Chapter 2

Representation via Subitization

2.1 History

2.1.1 *Learning to Count Objects in Images* (Lempitsky and Zisserman)

[LZ] uses the idea of density in order to count objects. Density is the idea that each pixel has the ability to change the total count by a certain amount by just existing within the image. In other words, the density is the rate of change over a single pixel. The density of every pixel forms a *density map*, and the way one is to use this density map is by calculating the integral of the density from the first pixel to the last.

The task then is to find a model that outputs a proper density map. As this paper points out, the main question becomes ground truth collection. They point out that when humans count, it is common for them to tap on each instance of the object in the image. They therefore decided that *dot annotations*, where the annotator puts a dot on each of the objects, is a reasonable expectation. However, dot annotations don't explicitly state density, just where the objects are. Therefore, the ground truth density needs to be modeled some other way.

Naturally, they needed to pick a loss function. They define a function called the *Maximum Excess Over Subarrays* (MESA). MESA finds the subarray (over x and y, forming a box) whose L_1 distance between two sets of density maps (in this case) is greatest. They give two reasons, via two counterexamples, for this choice. One of them is that, if we take the degenerate case of MESA where the only subarray is the whole image, then regression may be possible, but it requires that each sample be a full image (and also eliminates the usefulness of having a density map in the first place; they state that this is "a direct mapping from some global image characteristics...to the number of objects is learned"[LZ][§ 1.1]). This is in contrast to the other scenario that they mention, where

the summation of distances between each density value and its counterpart in the other density map addresses the issue of accurate density maps in theory and gives you substantially more samples to work with (as each pixel is a sample). However, it may be the case that an input image during training may output a density map that integrates to the right value, but small deviations between the ground truth density map and the predicted density map would cause the pixel-wise error to dramatically increase. They point out that this is bad for training purposes as it does not truly encode the counting error.

They settled on the ground truth density being modeled as a 2D Gaussian kernel. The Gaussian kernel's values were that of a normal distribution, so integrating over them summed approximately to 1, the number of objects covered by the kernel. The reason why the shape of the density does not matter, as they state, is that, in the end, they only care about the densities summing to the count of objects in the image. However, as stated previously, training a model to be specific about the density of each pixel allows for less training data.

For the first dataset that they discuss (involving determining the number of biological cells in an image), each annotation was given a kernel the size of one pixel, and that pixel's value was one as a result. The second dataset fo which they were counting humans, each annotation had a kernel with diagonal covariance matrix with 4s on the diagonal, making the kernel much wider than the former one. These sizes were chosen empirically. The regression model used in both was a simple linear one, whose inputs were specific features, not merely the values of the pixels in the image. Overall, the method was successful at outperforming what they deemed "baseline approaches"[LZ][§ 3].

2.1.2 *Counting in the Wild* (Arteta et. al)

[ALZ16] extends the findings of [LZ] (its density map and more) while considering a new annotation scenario. In this paper, they had a dataset of images of groups of penguins; however, mutiple dots instead of a single dot were placed on each penguin, one for each annotator that annotated the

image. Therefore, they posited that they can use this information to determine the size of each penguin in addition to other features.

Instead of training a density model based off a ground truth density map, they took an intermediate step. Specifically, they trained the network to segment the penguins from the background, and then used this segmentation to find connected components within the segmentation area. This allowed them to assign density to each connected component (instead of, say, one density for the whole foreground segment) resulting in the ground truth used to train the part of neural network dedicated to density estimation. They state that segmentation, with more importance placed on the network's respective segmentation error, will encourage the network to learn better representations, potentially leading to better density prediction. Another reason could be possibly related to the general sentiment of locational accuracy given in [LZ]. For [ALZ16], being locationally accurate can be justified by considering a much different scenario. In this case, one large connected component would have the correct density for integration, but would not encode the fact that pixels in the component that represent more area in the real world should probably be assigned higher density (as these pixels would be shortchanged with respect to ground truth density). However, it is important to point out that neural networks (one is used in the paper) might be able to learn this encoding by choosing a proper weight for the pixel location. On the other hand, a neural network trained with better information would likely perform better, so this argument likely holds water regardless.

Another output of the neural network was trained to be part of estimating the variance in the expected number of dots that would be placed in a connected component if the image were part of the annotated set of images. The variance predicted is spread out over the pixels so that each pixel gets a share of the variance. As a result, integrating the variance estimated each pixel over the connected component, just as is done with density, would get one the output actually desired.

The main finding of this paper is that, if there are multiple annotators per penguin, the distri-

bution of the locations of annotations within that penguin will depend on the penguin's size. As a result, it would make sense to take that into consideration when determining the correct density in a given area to train against. In order to compare the effectiveness of using the annotations to determine size, they ran this training method against two other techniques. Both the proposed method and one of the competing methods used each annotation as the center of a Gaussian distribution. Each annotation contributed an amount of ground truth to a pixel's label based on how far away it was from the pixel, with distance weighted by the pixel's respective value within the Gaussian kernel. The only differences were that the second method did not use spacing of annotations to determine an appropriate size for these Gaussian kernels, and that this method did not use segmentation to come up with a more accurate ground truth density map. This method had per-pixel depth information with the rest of the ground truth data.

While not as good, using the relative positions of annotations with segmentation worked almost as well using the true depth at each pixel in the positions's places, and substantially better than the third method.

2.1.3 *Salient Object Subitizing* (Zhang et. al)

From the perspective of subitizing, [Zha+15] took a more canonical approach. In this paper, they worked with the definition that subitization capabilities generally end for more than four objects within the image. With that in mind, they built a network (among other methods to compare against) that placed an image within five categories: counts zero through three were four of them, and a final "4+" category was used for images with more than three objects (the number, they state, comes from [ACF76; MS82] (in that order), but we too have heard from [ACF76] and, initially, [Ner14]). Further, they used the defintion of subitization that states that any kind of object within the image is part of the count (the objects do not need to be similar) as long as they are considered "salient" in the sense that they are important. They ended up addressing two tasks:

pure subitization and using the subitization count to make object detection more reliable.

They generated their own dataset by taking images from other ones and paid annotators via Amazon Mechanical Turk (AMT) [Ama] to determine how many objects were in each image. Further, they needed subitization performance to compare to; the AMT annotators were not subitizing when annotating, as the authors were looking for ground truth information. Although the author believes this was not stated in the paper, doing a human experiment via AMT is probably not easy to run. As a result, they set up an experiment where they asked three people separate from AMT to perform subitization, showing a participant each image for a half of a second. This was used as a point-of-comparison as subitization is an anthropological phenomenon. Further, they came up with various subitization algorithms, with their main contribution being two neural networks, one which directly regressed the counts, and the other which was used to generate features which were classified with a support vector machine. All but one of the other methods also used an SVM to classify method-specific features. Both network-based solutions handily beat the others, and the non-SVM network version drastically outperformed the SVM one except in detecting when no salient objects are present. They went on to apply the winner to two different detection problems, normal detection (where the network was used as a prior to determine whether any object should be detected at all) and object proposal (where the number of bounding boxes was limited by the count of objects determined by the network).

2.1.4 *Understanding the Ability of Deep Neural Networks to Count Connected Components in Images* (Guan and Loew)

[GL21], in the context of neural networks, posed the subitization problem one involving the counting of connected components. To look into this issue, a network was first trained on random pixel images where the pixel could only have the values of 0 and 1. The 1 was the "on" state, with "0" being off; the loss was some kind of measure between the number of pixels guessed and the

number of pixels that were actually in the image. It is not clear why this task was chosen, but the counts that the network output were nearly exactly correct. Afterwards, they counted groups of pixels as well as shapes. In the former case, pixels were set as on or off randomly, which created connected components formed by adjacent pixels that happen to be in the "on" state. The more of these neighboring pixels (meaning left, right, top, and bottom pixels) that are "on", the bigger the connected components. The types of shapes that were considered were triangles and circles.

The results lead them to claim "no matter how...the training and test sets [are designed], once the sizes or types of objects in tests sets become different from training sets, the predictions on test sets become incorrect" [GL21][§ III]. There were two different scenarios that they were referring to in this statement. In the first case, how big or how small a single shape ended up within the same image was independently and identically distributed (i.i.d.). This indeed had shown problems with the representation learned; see figure 2.1.4. While it did predict triangle counts from the same triangle generation parameters successfully, not only did different sizes throw the network off, but so did different shapes. They suspected that the average size of a shape being the same between images can result in the network learning that the number of pixels in the image (not clear if they mean "on" or, instead, "off" pixels) is the best feature to regress. As a result, they fixed the size of the shapes within a single image, and trained from those kinds of samples. They claim that this did not work either, but graph b of figure 2.1.4 actually does show some success, at least for triangles of vastly smaller sizes. Even for a small number of circles, there does seem to be some sort of subitization capability. In addition to this evidence, the prediction of the number of connected components formed by the random pixel-based images were not predicted poorly, either (however, they point out that the success of this network depended on the requirement that the training set be similar in possible connected components counts to the test set, which did seem to be the case; see [GL21][figure 5]). They go on to give rationales for the perceived failures of the network that this author does not necessarily believe to be true given that their networks did have some amount

of success within the definition [Zha+15] of subitization.

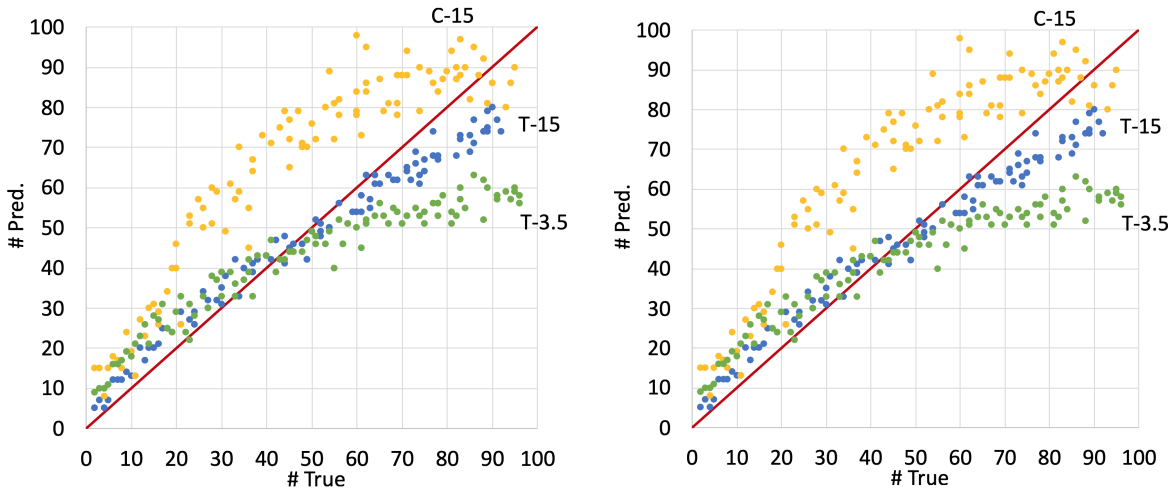

Figure 2.1: [GL21][figures 4 and 7]'s performance graphs of the model [GL21] chose. Letters on graph grids refer to the first letter of shapes triangle and circle, and the number after it is some measure of size.

We also refer to and discuss parts of this paper in section 2.2.

2.1.5 *Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps* (Simonyan, Vedaldi, and Zisserman)

One requirement of our methods specified in 2.2 is to be able to understand the underlying representation the network ended up developing. Therefore, we present [SVZ14], which does not specifically explore counting, but will give us the tools we need for our work.

[SVZ14] makes important contributions in two ways. The first is that they construct a method by which a network can, in a sense, paint a picture that shows what the network thinks a certain class looks like. They make the point that this method only describes the network, not how the network comes to its conclusion on a per-sample basis. To do the latter task, they describe a method of generating a saliency map, which we find very useful for our contributions.

Figure 2.2: Some images and their respective saliencies they generated; from [SVZ14][figure 2]

In the general sense, a saliency map is an image where each pixel is given an intensity value representing its contribution to the classification. In this paper, they work with the idea that gradients w.r.t. an image is a direct saliency representation; backpropagation starts from the desired class. The reasoning is intuitive: if the goal of a saliency map is to point out the most salient parts of an image, then one can think of large gradients as essentially exactly what we need. This is their method, but with a small modification: because each subpixel is assigned a gradient, they merge the three as but there is only one value assigned to each pixel in the saliency map. Merging consists of finding the channel with the greatest magnitude, and assign that magnitude to the pixel for which we need to generate a value. They perform some sort of normalization as the final values could be larger than 255 (or whatever limit the bit depth of the image implies). Figure 2.2 reproduces saliency maps that can be found in the paper.

Interestingly, they find that this saliency map can be used to determine which parts of the image are foreground and background. This is due to the fact that the salient pixels are indications of the foreground, and so they can be fed to an algorithm that performs the segmentation (they

use something in [BJ01] to do the segmentation, but this author does not know the details of that algorithm to elaborate how it works with the saliency map).

2.2 The Network Whisperer

We tackle the same problem as is found in [Zha+15; GL21] of subitization. However, there are a few differences between this implementation and each of those implementations. In contrast to [Zha+15], they work with a much wider definition of subitization, in which the counter can count every kind of object; the domain is not restricted to objects of the same type or roughly of the same type. We use a dataset that is extremely trivial (uniform shapes in an image with a black backgroud) relative to their dataset. Though ambiguous according to [Dic], we believe that theirs is the correct interpretation, especially from personal experience. Therefore, their problem is more difficult in this respect. However, they stick with the common mindset of four objects being the maximum subitization capability, and so compare their results to that limit (again, with a single category that pertains to images with four or more objects). We do not believe this is a thorough test of the capabilities of neural networks, and (though this fact was not a motivation for us) we take a stab at the neural network learning to count an arbitrary number of objects. However, "arbitrary" does not mean boundless as there can only be so many shapes within an image (a strong upper bound would be the number of pixels in the image, but that is not a supremum). Both [Zha+15] and our work attempt to detect objects using a classifier trained to count. However, our method performs the localization as a byproduct of the second derivative of counting, which, although not exactly clear due to time limits we had and ambiguity in the paper, differs from their methods; one of them uses the output of the model as a guide for a separate counting method, for example.

With respect to [GL21], they make much stronger claims that this author feels comfortable with. We aim to show that these networks can indeed subitize (up to a object-dissimilarity limit),

regardless of whether or not it is terribly accurate. Further, they pose this problem as determining connected components, which allows them to make a claim that one connecting pixel should be considered as merging two components into one. We think this is a dubious argument, mostly because that for the example that they show [GL21][figure 12], the vast majority of people would not likely be able to see the pixel-wide connection, and therefore four separate components would have been the guess by most. Coincidentally, we also happen to look at shapes, although we try to mix in different kinds and sizes of shapes into the same image; they do try different sizes, but the sampling process is different (which we will use in our implementation as a potential solution to poor saliency output).

The definition of our problem is as follows: given an image and a network, can we train the network to successfully count shapes in a subitization fashion? The shapes were automatically generated with random positioning; shape overlap detection was performed with the recommended[rMD18] Shapely[Gil+–] package, and any overlap rejected that shape's location. Two possible shapes, triangles and squares[1] can be generated. Images were drawn and saved with Pillow [Cm]. Larger shapes resulted in proper learning; we had no luck with smaller shapes. The network is a convolutional network, with no other layers besides activation functions and a linear layer at the end to output a single ReLU[GBB11][2]-filtered number. This is done because the network [Sze+14a] they use in [Mun+16] does as well (except for the output layer). The loss function is mean squared error, with the ground truth being the raw number of each kind of shape (this is in contrast to [Zha+15], which had separate classes that represented different counts). Because the images are automatically generated, we are able to use extremely large datasets, which may not be a reasonable expectation when training on non-artificial data. For now, we stick with shapes as it is a simplified problem for which a proof-of-concept could lead to further research into whether the concept can be applied in the real world (not just from a subitization standpoint, but also from

[1] This may be Professor David Crandall's idea

[2] This reference is commonly pointed to as the source of ReLU

a detection one as well). We used Chainer [Tok+19] to implement our research in this section, as well as CuPy [Oku+17], NumPy [Har+20], and Python 3 [Pyt]. There was previous code that addressed this topic based on PyTorch [Pas+19]; we did use it to some extent, but it was rewritten to use Chainer later on. This code may have used Nesterov momentum instead.

2.2.1 Detecting Shapes

To perform the detection step, a trained network[3] and an image are used to generate a saliency map via the method described in [SVZ14]. However, instead of backpropagating from the correct class's output (there is no concept of "class" in our network), we backpropagate from the addition of all the outputs (which ideally would give us the total number of shapes), one output squares and one for triangles. This addition occurs (most likely; thank the author's short memory) because the aim is to detect *all* shapes within the image, not just one type; however, it would be good to see if saliency for a single shape type could be used (depending on the success of the original method, of course).

The detection step comes in by exploiting the (yet unproven) property that the gradient of one pixel depends on the value of another. Ideally, if the derivative of the saliency of the pixel w.r.t. another pixel changes, then that implies that the changing value of the latter pixel (for this example, it decreases to become more like the background) causes the former to take on more of a representational burden in order to keep the object count up. However, it is possible that every pixel in the image can change the second derivative of the target pixel. To see why, let's look at the first derivative w.r.t. the image (f is the sum over the estimated shape count, x_i is subpixel i of image x, and a_k is the output of layer k for image x):

$$\frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial a_k} \frac{\partial a_k}{\partial x_i}$$

[3] trained with small batch sizes as we found this useful for some reason; it could be that small batch sizes increase variance, which results in kind of a simulated annealing-like behavior

If we take the partial derivative of $\frac{\partial f}{\partial x_i}$ w.r.t. subpixel j, we see that the product rule gives us non-zero values for some or all parts of the domain following what is shown in section 3.2.1; the difference here is that we don't use a non-linear loss function, so at least one activation function for this network *must* be non-linear (even if the function is piecewise, as described in 3.2.1) for this to be true. We do not go over why not all layers have to have a non-zero second derivative in that section (although work on the topic of 3.2.1 likely preceded, and therefore was the foundation of, this the idea of the type of activation causing second-order issues), so we will do that here. Here is the analytical derivation of the second derivative:

$$\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\frac{\partial a_3}{\partial a_2}\frac{\partial a_2}{\partial a_1}\frac{\partial a_1}{\partial x_i} =$$

$$\frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\left(\frac{\partial}{\partial x_j}\frac{\partial f_3}{\partial x_i}\right)$$

$$= \frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\left[\frac{\partial f_2}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_3}{\partial a_2}\right) + \frac{\partial a_3}{\partial a_2}\left(\frac{\partial}{\partial x_j}\frac{\partial f_2}{\partial x_i}\right)\right]$$

$$= \frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\left[\frac{\partial f_2}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_3}{\partial a_2}\right) + \frac{\partial a_3}{\partial a_2}\left[\frac{\partial f_1}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_2}{\partial a_1}\right) + \frac{\partial a_2}{\partial a_1}\left(\frac{\partial}{\partial x_j}\frac{\partial f_1}{\partial x_i}\right)\right]\right]$$

$$= \frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\left[\frac{\partial f_2}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_3}{\partial a_2}\right) + \frac{\partial a_3}{\partial a_2}\left[\frac{\partial f_1}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_2}{\partial a_1}\right) + \frac{\partial a_2}{\partial a_1}\left(\frac{\partial}{\partial x_j}\frac{\partial a_1}{\partial x_i}\right)\right]\right]$$

We can simplify this rather complex series of additions, and this looks like

$$\frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\left[\frac{\partial f_2}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_3}{\partial a_2}\right) + \frac{\partial a_3}{\partial a_2}\left[\frac{\partial f_1}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_2}{\partial a_1}\right) + \frac{\partial a_2}{\partial a_1}\left(\frac{\partial}{\partial x_j}\frac{\partial a_1}{\partial x_i}\right)\right]\right]$$

$$= \frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\left[\frac{\partial f_2}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_3}{\partial a_2}\right) + \frac{\partial a_3}{\partial a_2}\frac{\partial f_1}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_2}{\partial a_1}\right) + \frac{\partial a_3}{\partial a_2}\frac{\partial a_2}{\partial a_1}\left(\frac{\partial}{\partial x_j}\frac{\partial a_1}{\partial x_i}\right)\right]$$

$$= \frac{\partial f_3}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right) + \frac{\partial a_4}{\partial a_3}\frac{\partial f_2}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_3}{\partial a_2}\right) + \frac{\partial a_4}{\partial a_3}\frac{\partial a_3}{\partial a_2}\frac{\partial f_1}{\partial x_i}\left(\frac{\partial}{\partial x_j}\frac{\partial a_2}{\partial a_1}\right) + \frac{\partial a_4}{\partial a_3}\frac{\partial a_3}{\partial a_2}\frac{\partial a_2}{\partial a_1}\left(\frac{\partial}{\partial x_j}\frac{\partial a_1}{\partial x_i}\right)$$

We can expand this process to any number of layers. However, if each activation function has a second derivative of 0, the whole equation collapses to 0; we discuss why this is relevant and a problem shortly. If the last activation function's gradient has a gradient that is non-zero, then we

cannot guarantee the network's complete second derivative will end up being zero at some subpixel x_j. This is essentially the same point that 3.2.1 makes (and it may have been based on it too). This conclusion requires that the derivatives of the derivatives of an activation with respect to another evaluate (for example, $\left(\frac{\partial}{\partial x_j}\frac{\partial a_4}{\partial a_3}\right)$) to zero. This happens to be the case because the second derivative between two layers is zero, and the left-over derivative of the incoming set of activations w.r.t. x_j due to the chain rule application are nullified because it is multiplied by the aforementioned second derivative. So, avoiding zero-valued second derivatives is necessary (one non-zero second derivative will possibly result in a non-zero, end-to-end second derivative because every addition in the above equation could give us something non-zero)[4].

2.2.2 Algorithm

Now that we have established the basic concepts and requirements for our localization procedure, we can outline ways in which detection may happen. A way that we attempted (but were never able to fully iron-out) and inspired by the thresholding used in [Ree21][3:58] is that we take the highest saliency [SVZ14] value (which corresponds to a certain pixel i), and then calculate the second derivative of that value w.r.t. each pixel within the image (forming a saliency map in the same way that [SVZ14] does, but using the second derivative — this was done to ensure that second derivative values do not become negative when formulated as saliency due to the absolute value used in [SVZ14], and also to merge the three channels of derivatives that would result under normal partial derivative calculation of the first saliency map; this is why [SVZ14] takes these steps). We choose to backpropagate from the most salient pixel as, intuitively, that is the pixel most likely to be on a shape. We then threshold the second-derivative using some kind of intelligently chosen value, giving us a set of pixels that hopefully are related enough to the original pixel. Using this set, it is possible to draw a bounding-box around them, giving us detection. After this, we remove

[4]Stefan Lee confirmed my math (at least for ReLU) as he also believed that $\frac{\partial^2 f}{\partial x_i}$ is required to be non-zero

these pixels (including the one from which we backpropagated) from the set of unassigned pixels. We do this for each i (where i is from the remaining pixels) until we no longer have no more shapes to find (based on the count output by the network).

Unfortunately, we did not find this method reliable. For starters, the second derivatives only covered, at most, about half of the shape, which would leave us with an incomplete bounding box. This could be due to a poorly-chosen threshold value, but we did not have enough time to try other values. However, we also noticed that a lot of the selected second-derivative pixels were between two shapes, which obviously would cause poor localization. This is actually not that surprising; remember, the first derivative was the derivative for the predicted count. If a pixel in-between is chosen as i, then it likely influence the part of the total count that corresponds to the two shapes; as a result, it would not be surprising that pixels local to i would determine i's first derivative (say, if a new shape started to form using those children pixels). It is important to note that the case before this could be because these shapes were further away than others. However, as one can see in figure 2.4, the spatial breadth of the selected pixels is not nearly the same, so we may be able to make the claim that we did anyway. Another issue is that, as the loop continued, fewer shape-like structures formed by the second derivative were selected; many of the points were extremely local. A more intelligent pixel selection method would likely improve this method substantially. Regardless, we definitely see promising results (see figure 2.4) that make it clear (barring any missed strange scenarios that allow these images) that the network is probably learning some form of representation, especially because figure 2.3 show that the most saliency is being placed on the edges and the corners, what are likely the only features that these shapes have. The original image and its saliency can be seen in figure 2.3. Note that all experimental saliency pictures, including those for the saliencies of the saliency, have slight tweaks to them: we have taken the 10th root of each saliency output pixel so that the extrema are closer together, thus improving image readability, and they were normalized for image output (this applies only to saliency maps,

not the thresholded images). Further, Pillow was used to transform this output to image files. This also applies to the saliency maps generated for each saliency pixel. This is inspired by plots that use the log scale on the y-axis.

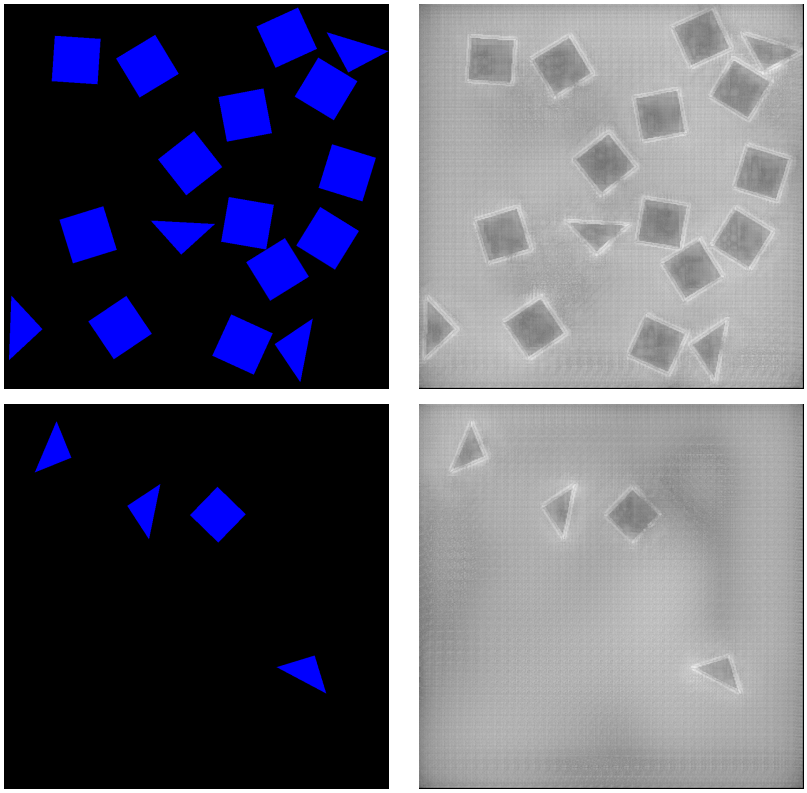

Figure 2.3: The first image is the image that the network sees, while the second image is the saliency for that image

2.2.3 Tweaks

Initially, our ReLU-activated network was trained on images of shapes that did not vary in size. Prediction was extremely accurate, so much so that it was suggested[5] that the network may be effectively adding up the number of black pixels (non-shape pixels) and dividing it by the number of pixels within a shape, or that corners were being counted instead (a graph of the network output against the ground truth created with [Hun07] when classifying both shapes and squares confirmed

[5]by the author's advisor, David Crandall

this). To ameliorate this issue, the network was trained on shapes of different sizes and was restructured to be bigger[6]. The ability of the network to predict correctly was significantly reduced but was not bad, initially indicating that the network had learned some kind of representation of the shapes. Further, it gave us saliency maps that appeared to be what we were looking for, which was that the network learned the representation of the shapes itself (see figure 2.5). However, due to the problem of the second derivative mentioned above, the second derivative did not give us anything meaningful. As a result, SoftPlus [Dug+01] was tried, but this resulted in numerical issues that sill might have squeaked some interesting detection pixels (see figure 2.6).

We moved to a sigmoid activation function (although we are not entirely sure why), with the network trained with Batch Renormalization [Iof17] (to get the sigmoid network to actually converge), but that only put the saliency in the blank space of the image. This is shown in figure 2.7. This remained a frustrating issue until we found [GL21], which states "[a]lthough sizes are random in a range, the average size...stabl[izes] [as] the number of objects increases" [GL21][page 3]. As a result, for two images that have the same counts, the number of relevant pixels will remain the same, which makes the network's job as easy as counting pixels. Therefore, we implemented the solution that they proposed, which was to use randomly-sized shapes, with all the shapes in the image taking on that random size. Size resampling happens after each image is generated. In [GL21], they allocated a static number of pixels to the image, and thus the size of the shape depended on how many shapes were to be put in that image. However, we made a slight tweak: we decoupled the number of shapes from limit on the number of pixels, and just made sure to pick a range of scales that allowed all the shapes to fit in a worst-case scenario (this was likely the problem they were trying to avoid by imposing the aforementioned limit). We decided to go this way because it seemed less likely that the neural network would find another non-optimal way to regress. The result from this method is discussed in the previous section.

[6]David Crandall implied that capacity might be a potential issue

Further tweaks need to address the problem discussed in section 2.2.2. Regarding the most salient pixels being between shapes, it may be necessary to modify the loss function from which to backpropagate. Candidates would need to encode encouraging strong gradients that *only* relate to decreasing the count by one. Another option is to threshold the first derivative saliency from above. As a result, errant pixels in the gap between shapes with large saliency would not be considered. Further, as stated previously, automatic determination of both thresholds would be substantially better than manual settings. This would become even more crucial when tackling real-world issues. An alternative solution may be to use K-means to group low, medium and high saliency pixels. The idea is that we would only stick with the medium-saliency pixels as the high saliency group would have many pixels that may change the count too much; rejecting those with low saliency values was justified previously. One may need to consider the distance metric to be used; for example, high saliency pixels may be distributed in an exponential fashion, so normal Euclidean distance would leave many undesired saliencies in the acceptable group.

Further, it is important to point out that real-world circumstances may actually provide *better, more reliable* saliency values. Counting shapes may have a significant flaw, which is that ordinary, monocolor shapes really don't have many features outside of their edges and corners[7]. If there are features that are internal to the object, it may result in cleaner saliency values that are focused on the object itself. However, it is also possible that the network learns to identify one component that many objects in the image share, and, as a result, its children pixels do not span the object entirely, making bounding box prediction problematic.

Our code and models can be found in the "Counting" folder at [Cut].

[7]David Crandall reminded me about corners possibly being important.

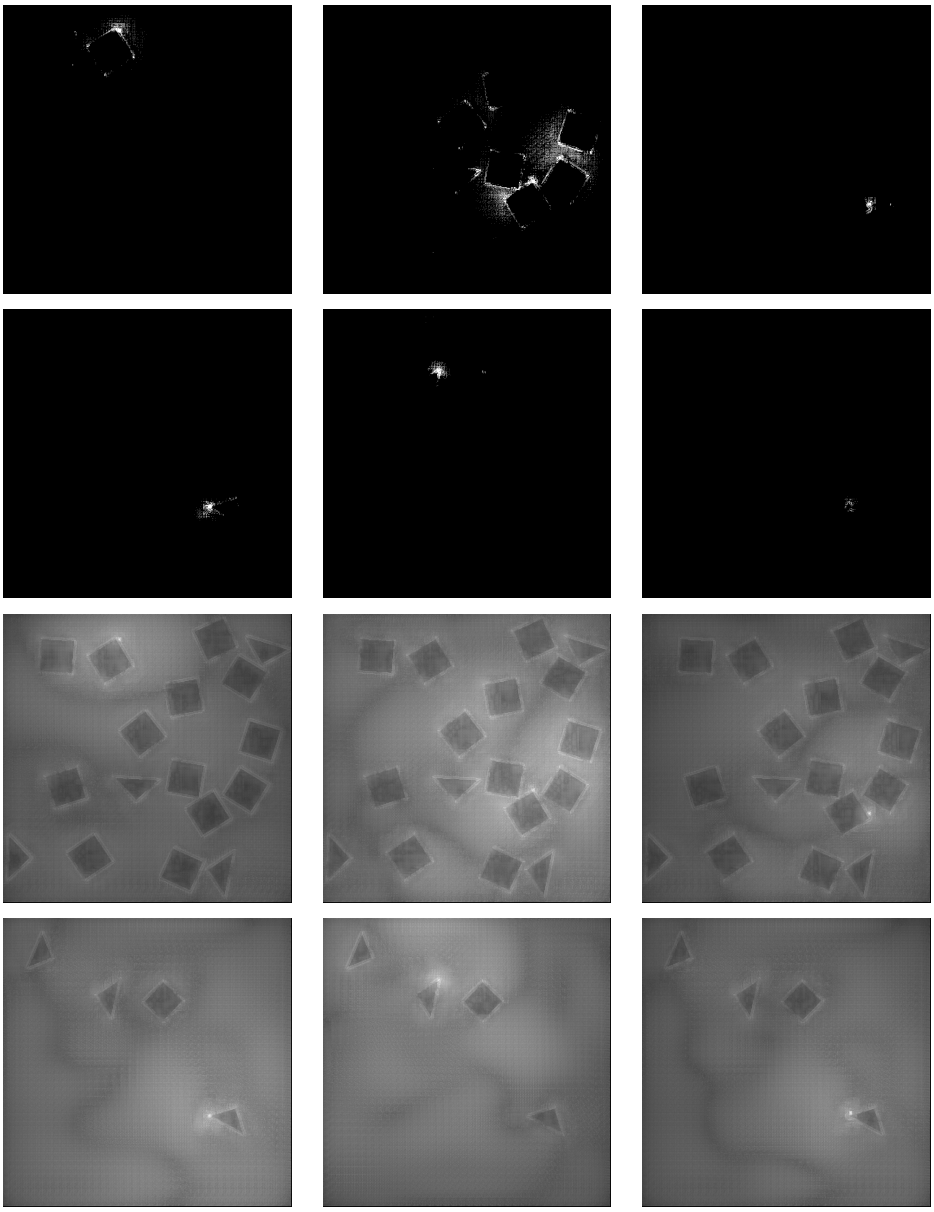

Figure 2.4: Each of the first two row represents a different image. The first images for these clearly shows the partial outline of a shape, while the second depicts the situation of a pixel being associated with multiple shapes. The third is the case where saliency selection derails and the area selected is not, or partially not, meaningful. Their full maps before thresholding are in the third and fourth rows.

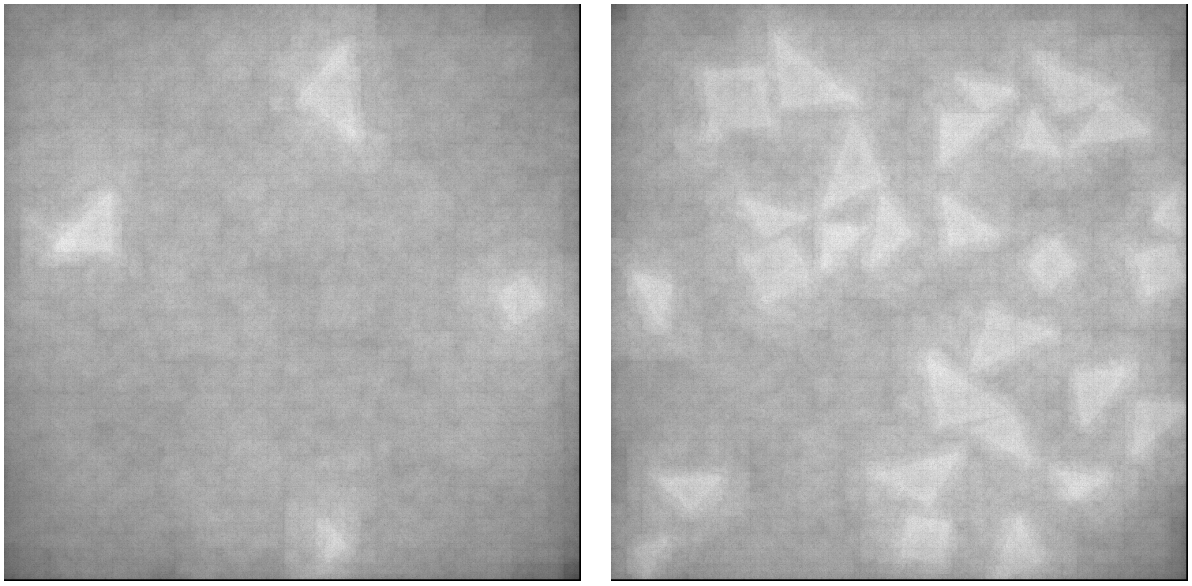

Figure 2.5: ReLU-based saliency.

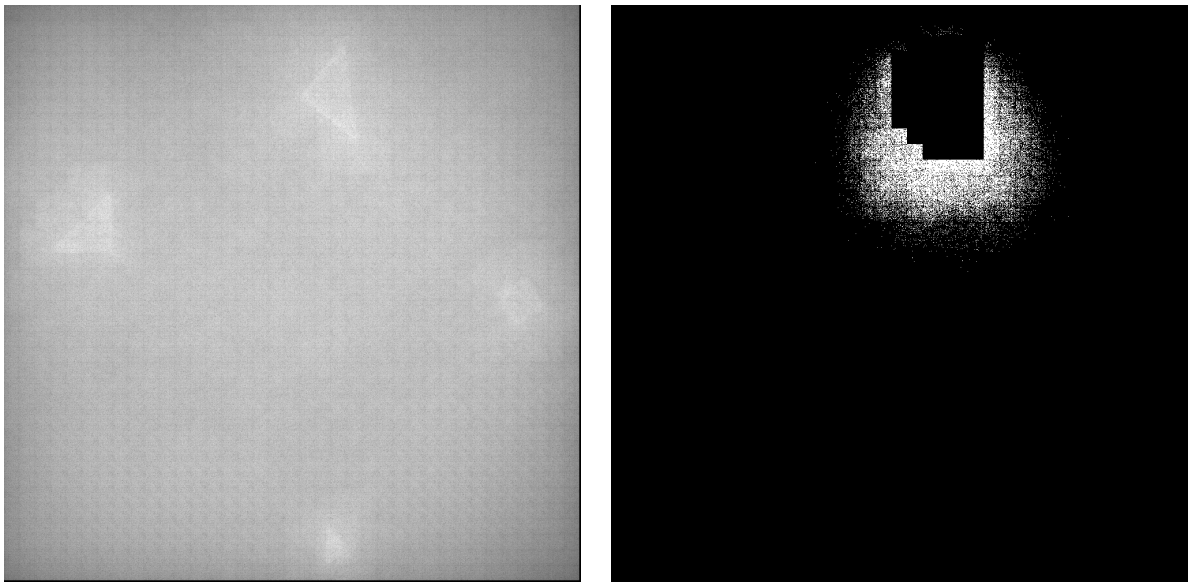

Figure 2.6: Our first attempt at using SoftPlus [Dug+01]

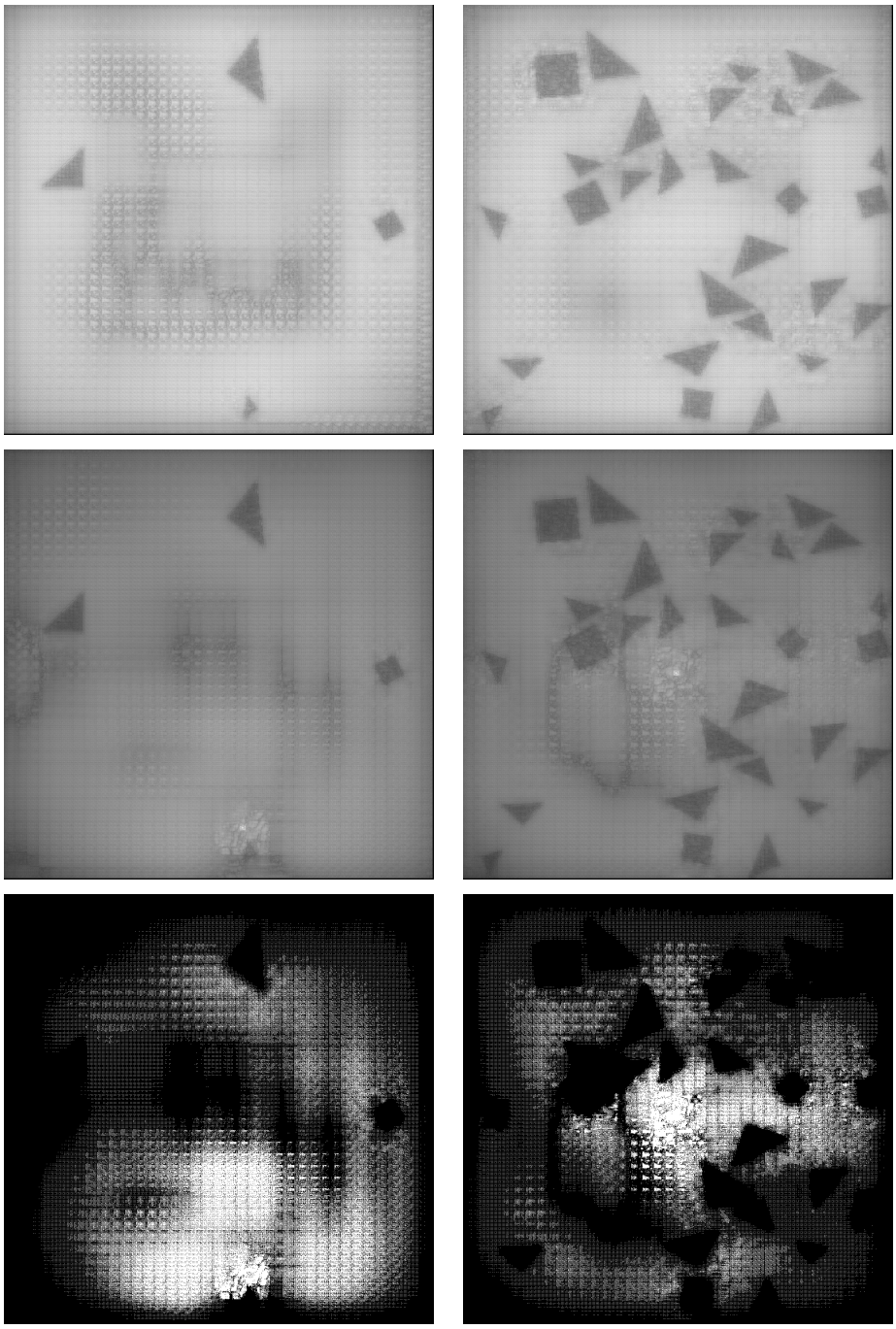

Figure 2.7: The first row shows saliency maps for two images, the second row shows the connection strength between the most salient pixel and others, and the third row are the pixels that "qualify" as being part of a shape

Chapter 3

Representation in the Face of Adversaries

3.1 Predecessors

3.1.1 *Intriguing Properties of Neural Networks* (Szegedy et. al)

The paper that truly brought to light the issue of adversarial attacks was [Sze+14b]. In this paper, they provide an optimization-based approach to generating what are known as "adversarial examples". To do this, they trained multiple models with fully-connected neurons on the MNIST training dataset[LCB]. One of the networks was an autoencoder. After performing this task, they generated adversarial examples for the sake of comparing the performance of the non-attacked network to the attacked network. In order to do this, they used L-BFGS[LN89] (modified to keep any changes within the L_∞ norm) to try to minimize the loss function, but with the target class being an incorrect class. They also made sure that the predicted class of the modified image is indeed the class they were hoping to achieve, and the modifications did not cause resulting input values to be less than zero or greater than one (the bounds on the possible values in the image). This resulted in images (derived from the training set) that visually still should not be misclassified, yet the neural network (and a few linear regression-based models) misclassified them. In fact, they were able to get accuracy down to 0% using noise that one would not think could change the classifer's output. These kinds of corrupted images were shown in their work, and are reproduced in Figure 3.1.

They also tried to use these adversarial images from one model on the other models. They did this for each model so that every model in the test was used to generate adversarial examples to be used on the others. The error rate for the other models was substantially lower; however, the minimum error rate (of all tests) was still a few times higher than data that had not been disturbed,

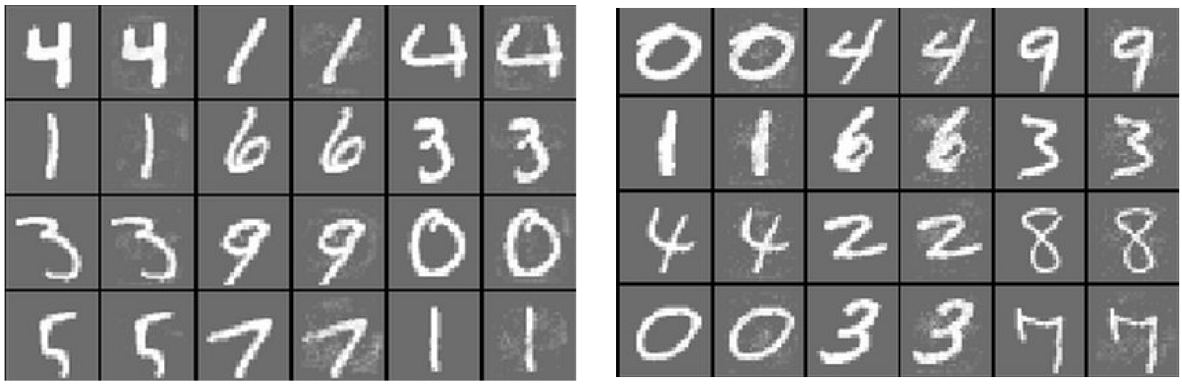

Figure 3.1: Adversarial images taken from [Sze+14b][figure 7] where the noisy images are the adversarial examples. See the figure of that paper for details about these images

with a few error rates well over 70 percent. Further, to eliminate the possibility that this isn't a issue of using the same dataset, one network was trained on a different subset of the MNIST dataset. For the two that were trained on the opposing subset, one of them had a different number of neurons than the other. They say these two differences were tested at once (with the different network) "to study the cumulative effect of changing the hyperparameters and the training sets at the same time"[Sze+14b][p. 8]. The results show that there is roughly an equivalent contribution, between dataset and model setup, to error rate with sabotaged images at low magnitudes. However, when they increased the magnitudes of the noise, the shared-training-scenario dataset caused much more misclassification when compared to having the training sets be the same when transferring in both directions between the different models.

The final part of the paper tried to analytically quantify the capability an adversarial sample has to affect the network's ability to classify things correctly. They used an interesting technique of a establishing series of derivative upper bounds that cascade through the network, resulting in a cumulative upper bound.

3.1.2 *Explaining and Harnessing Adversarial Examples* (Goodfellow, Schlens, and Szegedy)

[Sze+14b] was shortly followed up by [GSS15], which brought to light strong evidence that the cause of these adversarial examples is the linearity of the models being used (this linearity turns out to be limited, as discussed in 3.1.5). Mathematically, they point to the dot product that occurs in typical neurons and how drastically it can change when only small changes to the input vector in the dot product occur. Analytically, they come to this conclusion by pointing out that an adversarial example with noise η for original image x will end up in the dot product with weight vector w (note: all symbols used here are from the paper); further, we added this equation for clarity, but keeping w transposed instead of the input vector):

$$w^T(x + \eta)$$

After distribution, they point out that the second term in the resulting addition is $w^T\eta$, a non-negligible value (due to the number of elements participating in the dot product). They end up drawing this conclusion: the dot product determines the final activation value, so, under a max-norm constraint, noise added to a many-element vector may not appear to be much from the max-norm perspective, but actually *is* from the dot product perspective.

They bring up other theories about these phenomena, including the idea that nonlinearities are causing them and that "adversarial examples finely tile space like the rational numbers among the reals"[GSS15][p. 7]. To debunk the first notion, they developed their own attack called the *Fast Gradient Sign Method* (FGSM). This attack consists of optimizing an image by stepping away from the minimum, as opposed to towards it (the latter is the case in training, and the parameters being optimized are that of the network instead of the image). This uses the gradient (w.r.t. the image) of the actual loss of the image and its ground truth. However, a modification here is that the sign of the gradient, after being multiplied by an intensity-controlling value called ϵ, is used in place of

the gradient itself.

The goal of FGSM is two-fold. First, this method quickly generates adversarial examples instead of normal iterative optimization found in [Sze+14b], as they point out. They state that this makes training with adversarial examples (in addition to the regular images) much faster. The second reason for its existence is to prove that the attack space is a continuously linear subspace.

One reason that they note is simply because increasingly larger magnitudes of ϵ result in even higher confidence in the same wrong class. However, the series of pictures in figure 3.2 that shows a visualization of the examples at those ϵ values shows the vast majority of misclassified images being what they call "rubbish examples", and it appears that they use that term to describe significantly fewer samples in that image than we would. There are only a handful of misclassified subimages in that figure that, in our opinion, are actually recognizable enough to not be considered rubbish.

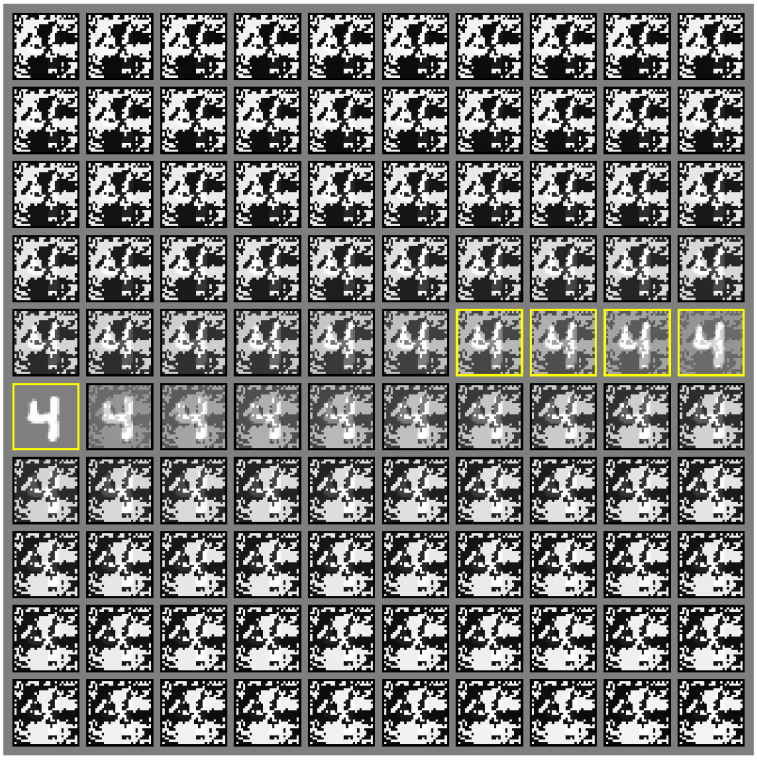

Figure 3.2: Part of figure 4 from [GSS15]. The number of rubbish examples appears to be more than they indicate.

A second thing that they point to is that adversarial examples created using a Radial Basis Function-based network and a shallow neural network that uses Softmax (likely just on the logits of the network) do not transfer as well to each other as the Softmax-based one does to a Maxout [Goo+13] network, which is more linear. A possibly important point to note is that they state that the first network is "shallow"[GSS15][p. 7], but this author is not sure exactly what that means in the context of Maxout networks.

3.1.3 *Towards Deep Neural Network Architectures Robust to Adversarial Examples* (Gu and Rigazio)

Prior to [GSS15], Shixiang Gu and Luca Rigazio, in [GR15], made an attempt at solving the adversarial example problem. They made two important contributions to this field.

The first was addressing the natural intuition that one would try to use an autoencoder to return an adversarial example to its clean state. Training of this autoencoder consisted of not just the inputs being the modified images and the original (off of which the loss is calculated) but also the non-modified image and the original. This was done to ensure that the autoencoder wouldn't unnecessarily de-noise its input. This was a largely successful method, but it revealed a flaw of using such techniques: this autoencoder is also a neural network, and it can also be attacked with essentially the same success rate of just attacking the classifier. Even worse, they found that the magnitude of this new noise was substantially *smaller* than the original noise. Further, they point out that "[f]or any pre-processing, it is always possible to backpropagate the error signal through the additional functions and find new adversarial examples..."[GR15][p. 5], essentially eliminating any usefulness of any other solutions like their autoencoder, and implying that differentiable methods of countering this noise, regardless if it is an autoencoder or some other method, are included too. It is important to point out, however, that these autoencoders are also linear neural networks, so it's possible that this mindset isn't valid for denoising approaches that avoid the linearity issue

discussed in [GSS15].

For their other contribution, they desired to introduce an additional penalty, during training, on the magnitude of the gradient w.r.t the image. The idea behind this is that small changes to an image should not escape the local, correctly-classified region created by training on the non-adversarial example. However, they stated that they did not have the computational resources to do this. Instead, for each layer, they penalized the 2-norm of the gradient from a layer's output w.r.t. its successor, using the image as a predecessor to the first layer. These penalties were added up per image (as in, the gradients penalized were obtained by backpropagating from the loss of each image, not from the loss of each batch). Their results did show improvement, but not in a significant way. One reason could be that the addition they tried does not obey the chain rule; multiplying the norms may have made more sense to get the chain rule effect. However, a more plausible explanation for this issue will be discussed later when we implement their ideal full-gradient ourselves.

3.1.4 *Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples* (Athalye, Carlini, and Wagner)

Unfortunately for the works that followed, [ACW18], for lack of a formal descriptive phrase, "rained on their parade". They confront the idea of *obfuscated gradients* (credited to [Pap+17] by [ACW18]), which they explain is the scenario in which there are no gradients that can be used for a white-box attack. To counter these defenses, to this author's best recollection, they employ two main methods, *Backward Pass Differentiable Approximation* and *Expectation Over Transform* (the latter is from [Ath+18]); for brevity, they use the acronyms *BPDA* and *EOT*, so we will do the same for the same reason.

BDPA solves the issue where the gradients are not easy to retrieve. In this technique, the part of the machine learning algorithm for which the gradients cannot be calculated is replaced by a near-equivalent that has this property. However, this replacement only occurs when calculating

gradients. They imply that it is likely necessary to use some kind of running average of the gradients used to generate the adversarial example when iterative methods are used; this is due to the fact that one single gradient calculation would be an approximation given the different function.

The next method, Expectation over Transform, finds the expectation of the gradient when any one of many different operations may be used on the input at any forward pass before it goes into the network. In the EOT paper [Ath+18], they highlight transformations such as scaling, brightness modification, and rotations. In [ACW18], these transformations take the form of anything non-deterministic that is done to the image for the intentional or unintentional sake of defending the network. The idea is that the average gradient will be a decent approximation to an adversarial gradient. It is not clear if the averaging only occurs in an offline setting or if it occurs online as well (such as during gradient descent optimization when creating the adversarial example). Further, in at least one case, the literal average is not used; instead, it is a simple summation of gradients.

They find the vast majority of their attacks successful implying, if not outright saying, that white-box attacks should not be the only metric by which a defense's effectiveness should be evaluated. Further, they outline a series of steps that a researcher should take to properly state the performance of their proposed defense; this appears to have mainly been done because most of the defenses evaluated were overstating their technique's effectiveness. These steps are 1. "[e]valuate against adaptive attacks"[ACW18][§ 6], 2. "[m]ake specific, testable claims"[ACW18][§ 6] and 3. "[d]efine a (realistic) threat model"[ACW18][§ 6]. The last two are self-explanatory, and for the first, "adaptive attacks" appears to be described as "[attacks] that [are] constructed after a defense has been completely specified, where the adversary takes advantage of knowledge of the defense and is only restricted by the threat model" [ACW18][§ 6.3]. In this case, the "threat model" defines what the adversary knows. In [ACW18][§ 6.3], they state that running the whole gamut of modern attacks would fit this definition.

3.1.5 *Towards Deep Learning Models Resistant to Adversarial Attacks* (Madry et. al)

[Mad+19]'s goal was to find both a defense and an attack mechanism that acted as some kind of an upper bound for their respective roles, at least from the standpoint of noise being within an L_∞ norm set of boundaries. They proposed adversarial training (which is when the network is trained on data that has been compromised) but a more advanced form of it. Adversarial training has been tried previously (most prominently, and probably originally, in [GSS15]). However, as demonstrated in [Mad+19], [GSS15]'s procedure is not a sufficient procedure to ensure that one's model can handle adversarial examples.

They start out their search by analytically encoding the adversarial training issue in the form of a loss function that, when this function's value is reduced, is equivalent to min-max optimization. In this scenario, the "max" is the adversary maximizing the loss function, while the "min" is the defender minimizing that maximum. The question they then pose is: what kind of adversary can actually find something at least close to the maximum loss when generating an advesarial example? It turns out that

1. doing repeated FGSM [GSS15] steps a certain number of times on the respective output of the last step, clipping to be within the allowed ball of noise, and then
2. trying that loop several times but starting at uniformly random locations within the L_{inf} ball of allowed noise around the image

reveals that the set of images generated by this method has little variance in loss value. They refer to this method as *Projected Gradient Descent* (PGD). The conclusion that they draw is that

> *...our exploration with PGD does not preclude the existence of some isolated maxima with much larger function value. However, our experiments suggest that such better local maxima are hard to find with first order methods...* – [Mad+19][section 3.2]

and that, as a result, PGD (with the aforementioned randomization and, to the best of this author's understanding, choosing the best example out of all of them) likely gives us something close to the

best attack possible.

Assuming that their assumptions are correct, this conclusion (alongside other point(s) made in the paper) completes the requirements needed to make adversarial training an effective way to thwart attacks. In order to empirically prove that adversarial training with PGD actually works, they tested three different things: a white-box attack and two black-box attacks, each of which were attempted using multiple attack methods, both on MNIST[LCB] and CIFAR-10[Kri09][ch. 3]. Both as a defense and as an attack, PGD dominated the other methods. For adversarial training, only one random point was selected, with their reasoning appearing to be that successive epochs will not change the model enough such that only attacking once per epoch over several epochs will result in roughly the same adversarial examples being generated using the full PGD attack per example during one epoch. In comparison to FGSM-based adversarial training (modified from [GSS15] by having the training set consist only of perturbed images), there were many instances in which, from the white-box perspective, any sort of robustness completely or almost completely disappeared when using the CIFAR-10 dataset. Black-box attacks caused a less dramatic decrease in model performance, but it was still significant. White-box attacks on models with PGD-based adversarial training were only mildly successful on MNIST using basically any attack; however, for CIFAR, "[t]he adversarial robustness of our network is significant, given the power of iterative adversaries, but still far from satisfactory" [Mad+19][page 12], nowhere near matching the MNIST network when attacked. Typically, attacking with PGD resulted in the lowest scores, and PGD with multiple candidate adversarial examples per image proved to be the most detrimental. There was only one scenario in which FGSM was more effective as an attacker, and that was in a black-box setting with a model from a different paper, [Tra+17]. To wrap up, they conclude that attacks like FGSM are not adequate by pointing out [Mad+19][§ 6] that the fact that adversarially-trained networks that use FGSM cannot resist PGD, and, in the same section, point to [Tra+17][1] as proof

[1]The author has not read this paper

that non-linear attacks may be more viable the further out the adversarial example is from the sample.

3.1.6 *Adversarial Logit Pairing* (Kannan, Kurakin, and Goodfellow)

[KKG18] came up with the idea of *Adversarial Logit Pairing* (ALP). This method puts both images with and without the perturbation through the network, but, instead of just making sure that the adversarial image predicts the same class as the normal one (a la [GSS15]), the L_2 distance between the logit outputs of both images is also penalized. In the case of a batch of images, the average L_2 norm is used.

They also attempted *Clean Logit Pairing* (CLP) which involves no adversarial examples. Instead, they did the same logit penalty between pairs of undisturbed images in the batch, but did not do adversarial training. Again, this penalty is averaged over every image in the batch. They point out that the effect of this is that the logits for each image will be more even due to the fact that the other image probably has substantially different logits, so the penalty will bring the two logits together. Because of this, they also tried a technique called *Logit Squeezing*. If the aforementioned cause of the effectiveness of Clean Logit Pairing is what is happening, then it may make sense to just penalize the norm of the logits of each image (instead of the distance between two different images). It is not stated, but each of these penalties (one per image) are probably also averaged as in the previous methods.

When compared to a modified version of the PGD defense that originated in [Mad+19], ALP ended up modestly beating PGD against both white and black box (transfer) attacks using PGD on MNIST[LCB] and Street View House Numbers (SVHN)[Net+11], and only lost by a small amount when testing clean examples from SVHN. It also performed best against other defenses on tests on ImageNet[Rus+15], both black-box and white-box, while non-adversarial performance remained on par with the PGD-based defense. Both CLP and Logit Squeezing were as good from a white-box

perspective (although not quite good as ALP), and was comparable in other contexts.

3.1.7 *Ensemble Adversarial Training: Attacks and Defenses* (Tramèr et. al)

[Tra+20] addressed the issue of black-box attacks. They found that the testing regime used previously did not adequately assess whether adversarially trained models can handle black-box adversarial examples in general. Specifically, they found that the linear black-box attacks used (for example, FGSM[GSS15]) were not good at reaching the maximum in the min-max problem discussed in [Mad+19], and that white box attacks turn out to be not as good as black-box attacks on models that have been trained with first-order, single step attacks. Further, it is possible that white-box attacks on these adversarially trained models transfer poorly when used as part of a black-box attack on *non-adversarial* models.

They came to these conclusions first by probing the loss function space with respect to a two-dimensional subspace whose axes are essentially the sign of the gradient and a vector perpendicular to it (likely inspired by [GSS15]) scaled by increasing adversarial perturbation magnitudes. At each point, two noise vectors generated from the first and second magnitudes multiplied by the sign of the gradient and its orthogonal counterpart, respectively, were added to the non-adversarial example (we will call these scaled vectors a and b, again, respectively). This generated a three-dimensional map, with the z axis representing the loss at the adversarial example represented by the point generated from the addition of a, b, and the original image (axes represented the L_∞ magnitude of a and b). See figure 3.3 for an example of a plot. They point out that the gradient at small perturbation values points in a drastically different direction compared to the direction that would lead to the largest increase in loss. They state that this explains both why white-box attacks on the aforementioned adversarially-trained models do not work well on both the adversarially-trained and non-adversarially trained models: this gradient does not represent the most effective direction for ascent. Therefore, testing in the signed direction gives a false sense of security w.r.t. the defended

model, and these new images also, as a result, do not perform well as a black-box attack against undefended ones. They call this phenomenon *gradient masking*.

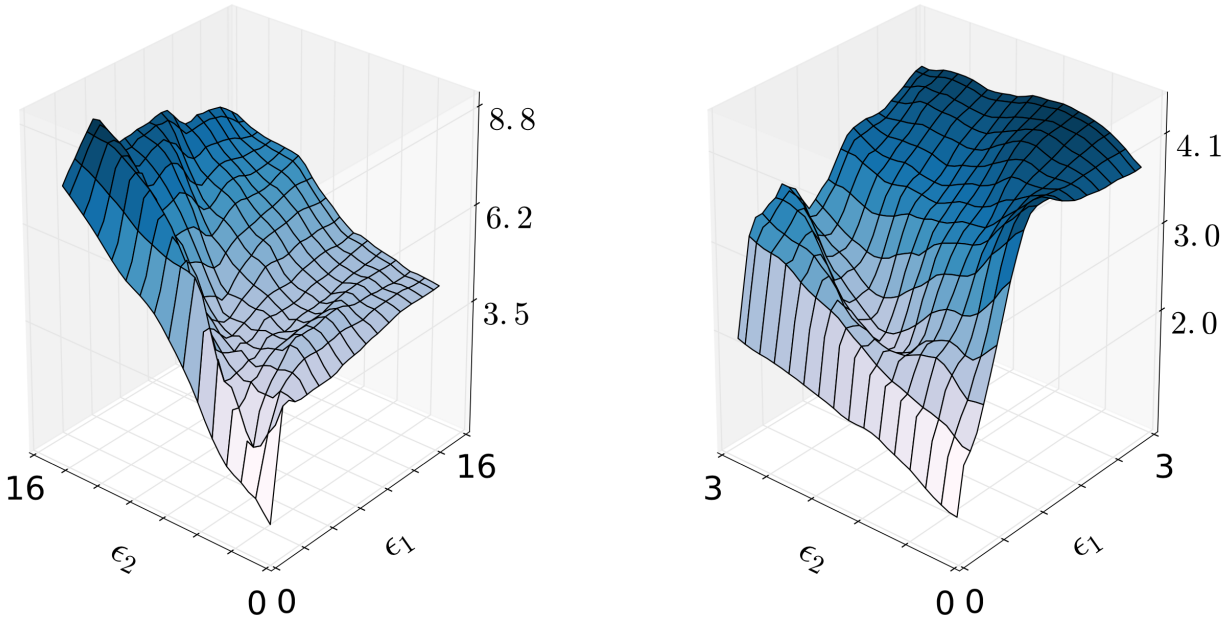

Figure 3.3: An example of gradient masking. The gradients in the data point's immediate vicinity point one direction, while the actual maximizer is in the other direction. Image is a snapshot from [Tra+20][figure 4].

To tackle this issue, they introduced *Ensemble Adversarial Training*. In this technique, not only do they train on white-box adversarial examples, but they also incorporate black-box examples generated from non-adversarially-trained models. Each model contributed the same number of adversarial examples, but only one model's adversarial examples were used each iteration. It appears that only adversarial examples were in the training set. They found that this defense had much better black-box resistance, and it eliminated cases in which the black-box attack was better than white-box.

3.2 Defense Proposals

3.2.1 Gradient Magnitude Reduction

One solution to the adversarial examples problem is a simple and intuitive one: to penalize the gradient with respect to the image. As stated in the literature review, this was proposed by [GR15] (however, we independently developed it). To see why this makes sense, consider the fundamental situation when it comes to adversarial examples: as [GSS15] states, dot products in deep learning can have *many* terms. If this is unavoidable (more on that later in other proposed defenses), then all we can do is to reduce the importance of each pixel/subpixel, which can be seen as penalizing the gradient w.r.t. the (sub)pixels.

How to do this is not necessarily straightforward. For starters, we could penalize its norm. Which norm to use may matter, but, in our implementation of this technique, we stuck with a normal 2-norm. The calculation of the loss comes in two steps. The first step is do normal backpropagation towards the image in order to get the gradient with respect to the image. The norm is then calculated, and added to the normal loss term. Backpropagation occurs from there, which requires calculating the "second derivative" of the gradient (we use the derivative of the norm as the the gradient is a vector and, therefore, its gradient is not well-defined). However, our choice of activation function(s) may be problematic if they are not smooth (for example, the ReLU[GBB11] activation function, which we found to be problematic)[2]. To see why let us look at the ReLU activation function. ReLU is specified as

$$\begin{cases} p & \text{for } p > 0 \\ 0 & \text{otherwise} \end{cases}$$

where p is the activation potential coming into the activation function. On one side of this piecewise function, the derivative is always 0, and on the other side, it is always 1. It is plain to see, then, that

[2]Double-checked by Stefan Lee

the second derivative will be zero everywhere, thus providing no utility when it comes to gradient descent. This is a flaw in the nature of its linear, piecewise behavior; mathematically, the second derivative does *approximately* exist in the form of the derivative of the SoftPlus [Dug+01] function. As a result (see § 2.2.1), the image's second derivative is zero. This is the case if we ignore the loss function, as one thing to consider is the loss function's effect on the second derivative. Here, we can run into some trouble in our hypothesis. Considering that using a smooth non-linear loss function L is extremely common (for example, mean squared error), there *does* exist a non-zero second derivative for it. When applying the product rule in this situation, for neural network f, we get

$$\frac{\partial}{\partial a}\left[\frac{\partial L}{\partial f}\cdot\frac{\partial f}{\partial a}\right]$$

as one of the terms in the product rule. This is non-zero (as our only hope of this being zero is if the *derivative* of the network output is zero). So, it *is* possible to get some second derivative out of a ReLU-based network (or any piecewise linear network, for that matter). However, for our experiments, smooth functions were used to be more mathematically sound and to be more sure that we have gotten closer to exhausting ways in which this method can go wrong.

Following [GSS15], we backpropagate from the correct class's loss. We will call this method *Gradient Magnitude Reduction* (GMR).

3.2.2 Elastic Sigmoid

A simple modification to penalizing the gradient is that, while doing so can change the weights, a sigmoid function's derivative formulation cannot. Therefore, it make make sense to modify the sigmoid activation function so that this is possible. There are many ways to do this: essentially, one could enumerate all different combinations of sigmoid and some learnable parameter. One simple use case for such a parameter is to multiply the activation potential by a scalar parameter s. The reason why this can help change the layer's derivative is very simple: given the chain rule, the

derivative w.r.t. the potential d (for "dot product") for the sigmoid activation function is

$$\begin{aligned}\frac{\partial \text{sigmoid}(d)}{\partial d} &= \frac{\partial}{\partial d}\frac{1}{1+e^{-d}} \\ &= -\frac{1}{(1+e^{-d})^2}\frac{\partial}{\partial d}e^{-d} \qquad (3.1) \\ &= -\frac{1}{(1+e^{-d})^2}e^{-d}\frac{\partial}{\partial d}(-d) = -\frac{1}{(1+e^{-d})^2}e^{-d}(-1)\end{aligned}$$

When we add in s, (3.1) (and the rest of the derivative) becomes

$$-\frac{1}{(1+e^{-sd})^2}\frac{\partial}{\partial d}e^{-sd} = -\frac{1}{(1+e^{-sd})^2}e^{-sd}\frac{\partial}{\partial d}(-sd) = -\frac{1}{(1+e^{-sd})^2}e^{-sd}(-s)$$

As you can see, s has a direct effect on the derivative of this new sigmoid function. If this is a learnable parameter, decreasing $|s|$ through gradient descent will help with decreasing the norm of the gradient. We call this new activation the *Elastic Sigmoid*.

Note that there are nice properties in both directions in which the magnitude of s can change. If said magnitude gets smaller, we get the aforementioned effect. However, it is possible that it becomes large if activation potentials never end up close to the origin. This is because, as s increases in magnitude, the sigmoid tightens up to approximate the step function (or the flipped-around-the-y-axis version of it caused by $s < 0$), exploiting the flatness of the step function as long as the potentials end up far from the d-axis origin.

Another decision to make is whether or not there should be an s for each neuron in the layer. This obviously allows more flexibility if one neuron could benefit from a differently shaped Elastic Sigmoid compared to another. We just use a single s for the layer's activation due to time constraints.

3.2.3 Pairwise Difference

The next defensive structure we introduce is partially in response to the claim in [GSS15] about dot products wildly changing as a result of how many components are multiplied within it. Further, it

is "inspired" (in the loosest sense, authoritatively speaking) by this author's very unscientific and personal experience when trying to decipher hard-to-understand objects, visually speaking.

Let's say that we have a group of pixels for which we need to decipher what it represents. In this case, the group takes on some kind of noise which makes it hard to recognize, yet the general themes are still contained within the group (for example, it might be a very dark patch that needs careful examinations of the relationship between connected components within it). Personally, the author tries to find these connected components and perform the comparison. So, with that in mind, we extend this technique to the deep learning case.

Inspired by convolutional neural networks, we take adjacent patches of pixels, and, in order to approximate the connected-components analogy, we simply compare each pixel (or in the three-channel case, subpixel, which we will refer to from here on) to every other within the patch. The way we chose to compare subpixels is via a modification of the Distance Matrix[var]. Instead of squaring the difference between subpixel values, we do not square at all. The reason for this is simple: in the standard image processing case, distance would result in the same output when one subpixel is less than or greater than the other used in the comparison. For this use case, that is not necessarily ideal: if one pair is the inverted version of another within the image, they would get the same distance value, but the inversion should really be considered a different relationship.

In mathematical terms, what follows is the definition of the difference matrix, D. Each subpixel at position j in the vector $\mathbf{v}$ that holds every element in the patch is subtracted from element i from that same vector. This creates the element D_{ij}; in other words, the index of the subpixel from which element j is being subtracted is i. Again, notice that this is derived from the definition of the distance matrix, except in that case, D_{ij} is actually $(\mathbf{v}_i - \mathbf{v}_j)^2$. See figure 3.4 for a graphical view of this neuron's layout.

The difference matrix alone does not actually solve the problem described in [GSS15]; this is because, if all these subtractions are used in a neuron by multiplying each subtraction by one of

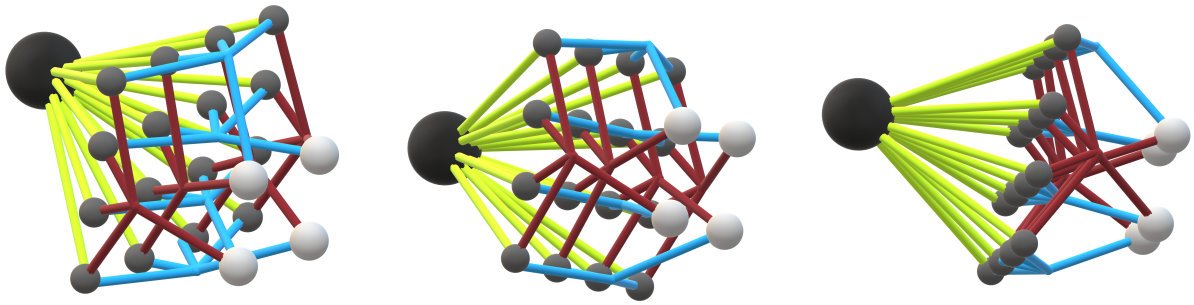

Figure 3.4: Visual of Pairwise Difference. From light to dark, the spheres represent the input, the subtaction + sigmoid, and the output of pairwise difference (sans the activation function). Burgundy lines indicate that the value it links from is being subtracted from the teal line's equivalent. Generated using Paint 3D [Mic].

the neuron's weights, noise added to a patch causes noise in the subtractions, which leaves us at the same issue (or possibly even worse given the squared number of elements, although we have not done the math to prove this claim). Further, we just want to know how the subpixels compare, not how they subtract. As a result, before the dot product of the neuron, we put each subtraction through a sigmoid activation function. At the first layer, if the input contains integer values (as is the common representation of subpixels), we won't see much change in the sigmoid in the face of noise as the range of values is typically 0 to 255, whereas the sigmoid activation function plateaus not far (relatively speaking) from the origin. This is the benefit of pairwise comparisons: small changes in a subpixel values do not have the same effect on the comparison (just between two subpixels) as it does on a dot product. To wrap up, the output neuron does not require any specific activation function, as most of the flatness comes from the preceding sigmoid functions. We call this neuron *Pairwise Difference*.

However, there is still a rather large elephant in the room: computational complexity is essentially squared, as we are building a pairwise matrix from a single vector. This is not a trivial

problem to address; for example, one might just use the non-zero values in the upper triangle of D because D is antisymmetric, so the information gain of keeping the lower triangle is (hopefully) very minimal, at least for D. However, this method suffers from not being able to treat the non-commutativity of subtraction differently, as there is no weight for the second permutation of the operands of subtraction. Further, effectively halving the number of inputs to the dot product only reduces the dot product complexity by a constant factor, so, theoretically, larger and larger kernels, from a practical standpoint, still might be out-of-reach. An alternative to the difference matrix might be that, as in the "mini-networks" of Inceptionv3[Sze+15], we do tiny dot products, and then take dot products of those tiny dot products, etc. Each tiny dot product would need to go through an activation (a pseudo-mimic of the difference matrix with just a few elements participating in the dot product) as, otherwise, the result of the "mini-network" would be very similar to a normal dot product from the perspective of [GSS15]. A third method would be to use this type of layer on just the image as most of the noise filtering will likely occur within that layer.

In addition to these computational issues, when programming such layers, it is very easy for memory complexity to grow as quickly as computational complexity. This is due to the fact that, in frameworks such as Chainer[Tok+19] and PyTorch[Pas+19], at the very least, it is easy to inadvertently suffer from holding the full difference matrix in memory. We can get around this by performing the subtraction, sigmoid, and weight multiplication operations one-at-a-time for each pair formed by a reference subpixel p_i with static index i and every other subpixel p_j in the kernel; if this happens for each static subpixel, we end up with a dot product for every reference pixel, which requires a single value of appropriate size in memory. Parallelization can still occur by doing this for each i simultaneously. Note that this does not differ from the memory requirements of holding each subpixel in memory, which is a tractable usage of memory. However, at the time of writing, this implementation had not been finished.

From a gradient standpoint, Pairwise Difference is a little complicated. Following the intuition

of [PP12][eqns. 136] where every path a gradient can take to a variable has a gradient that must by multiplied by the gradient from the start of the path to the variable (this is used in the multivariable chain rule, whose necessity was made clear to us by [LKX]["Gradients add up at forks"]), there are two types of paths for our specific problem where this may be the case:

1. the paths attributable to each image in the batch, and
2. the paths inside those paths that are associated with each spot to which the neuron was applied

Note that these are really just the same paths (all these paths fit case 2), but they are separated here to show that there is a hierarchy of paths, which may ease the understanding of the following derivation of Pariwise Distance's derivative.

For one patch p of one sample in the batch (that has been processed by layer i's neuron, n_i, contained in a Pairwise Difference layer), the output of $n_i(p)$ is a dot product between the sigmoid-filtered subtractions between each element r and element s in the patch and n_i's weights, w_{rs}. We will call "sig" the sigmoid activation function and "sub" subtraction formulated as $\text{operand}_1 - \text{operand}_2$. The derivative with respect to w_{rs} is

$$\frac{\partial n_i}{\partial w_{rs}} = \frac{\partial}{\partial w_{rs}} \sum_{a \in p} \sum_{b \in p} w_{ab} \cdot \text{sig}(\text{sub}(a, b)) = \frac{\partial}{\partial w_{rs}} w_{rs} \cdot \text{sig}(\text{sub}(a, b)) = \text{sig}(\text{sub}(a, b))$$

Each of the volumes (one for each sample in the batch) uses w_{rs} (abstracting away p so that it represents just the patch that would be taken out at p's location). According to the multivariable chain rule, for a loss function L, $\frac{\partial L}{\partial w_{rs}}$ is the summation of the partial derivatives calculated by multiplying $\frac{\partial L}{\partial n_i}$ by $\frac{\partial n_i}{\partial w_{rs}}$.

To keep backpropagation[RHW86] going to the previous layer, we also need to know $\frac{\partial n_i}{\partial r}$ (where, again, r is in p). In a fashion similar to w_i, r participates indirectly in multiple dot products against many different weights. Not only is r used in $\text{sig}(\text{sub}(r, r'))$ where r' is any element in p, but it is used in the other direction, $\text{sig}(\text{sub}(r', r))$. For the former, the derivative for each of these is

$$\frac{\partial \text{sig}}{\partial \text{sub}(r, r')} \frac{\partial \text{sub}(r, r')}{\partial r} = \frac{\partial \text{sig}}{\partial \text{sub}(r, r')}(1) = \frac{\partial \text{sig}}{\partial \text{sub}(r, r')}$$

and for the latter, it is

$$\frac{\partial \text{sig}}{\partial \text{sub}(r', r)} \frac{\partial \text{sub}(r', r)}{\partial r} = \frac{\partial \text{sig}}{\partial \text{sub}(r', r)}(-1) = -\frac{\partial \text{sig}}{\partial \text{sub}(r', r)}$$

Again, since the multivariable chain rule allows us to add all pairs of these derivatives together, for all patches P_r that contain r, we end up with the derivative being

$$\begin{aligned}
\sum_{p \in P_r} \sum_{n \in neurons} \frac{\partial n}{\partial r} &= \sum_{p \in P_r} \sum_{n \in neurons} \sum_{r' \in p} \left[\frac{\partial n(p)}{\partial \text{sub}(r, r')} \frac{\partial \text{sub}(r, r')}{\partial r} + \frac{\partial n(p)}{\partial \text{sub}(r', r)} \frac{\partial \text{sub}(r', r)}{\partial r} \right] \\
&= \sum_{p \in P_r} \sum_{n \in neurons} \sum_{r' \in p} \left[\frac{\partial n(p)}{\partial \text{sub}(r, r')}(1) + \frac{\partial n(p)}{\partial \text{sub}(r', r)}(-1) \right] \\
&= \sum_{p \in P_r} \sum_{n \in neurons} \sum_{r' \in p} \left[\frac{\partial n(p)}{\partial \text{sub}(r, r')} - \frac{\partial n(p)}{\partial \text{sub}(r', r)} \right] \quad (3.2)
\end{aligned}$$

The derivatives that remain in this equation are easy to derive based on the description of Pairwise Difference (a simple multivariable chain rule application), so we do not find it necessary to do so here.

3.2.4 Angular Neurons

Again, addressing the issue from [GSS15] with the dot product, another structural approach was taken. To reduce the impact of the number of elements in the input vector(s), the cosine between the input vector and the weights was considered. There is more than one way to view why this may address this issue. A goal of this layer is subtle: we want to make sure that extreme dimensionality does not increase the chances of the dot product going through major change. In the case of cosine, we can see why it may address this issue: generally (more on that later), as the number of elements increases, the number of elements that need to change in order to change the angle the same amount also increases. Therefore, if a very small angular neuron with, say, two inputs, does not change easily, the idea is that one with many inputs would not either.

Using the identity $\cos(\mathbf{a}, \mathbf{w}) \|\mathbf{w}\| \|\mathbf{a}\| = w \cdot a$, one can see that $\cos(\mathbf{a}, \mathbf{w}) = \frac{w \cdot a}{\|\mathbf{w}\| \|\mathbf{a}\|}$. Following standard neuron implementations, we add a linear bias term to the resulting cosine as well. To

see why a decreasing derivative is the case, consider the derivative of the cosine (we omit the bias because it disappears when differentiating due to it having no analytical relationship with $\mathbf{w}$ and $\mathbf{a}$)

$$
\begin{aligned}
&\frac{\partial}{\partial a}\cos(\mathbf{a},\mathbf{w}) \\
&= \frac{\partial}{\partial a}\frac{\mathbf{w}\cdot\mathbf{a}}{\|\mathbf{w}\|\,\|\mathbf{a}\|} \\
&= \frac{\|\mathbf{w}\|\,\|\mathbf{a}\|\,\frac{\partial}{\partial \mathbf{a}} w\cdot a - \mathbf{w}\cdot\mathbf{a}\frac{\partial}{\partial \mathbf{a}}\left(\|\mathbf{w}\|\,\|\mathbf{a}\|\right)}{\left(\|\mathbf{w}\|\,\|\mathbf{a}\|\right)^2} && \text{quotient rule} \\
&= \frac{\|\mathbf{w}\|\,\|\mathbf{a}\|\,\mathbf{w} + \mathbf{w}\cdot\mathbf{a}\,\|\mathbf{w}\|\,\mathbf{a}}{\|\mathbf{w}\|^2\,\|\mathbf{a}\|^3} && \text{derivative of } \mathbf{w}\cdot\mathbf{a} \text{ and } \|\mathbf{a}\|^2 \text{ are } \mathbf{w} \text{ [PP12][eqn. 69] and } 2\mathbf{a} \text{ [PP12][eqn. 131]} \\
&= \frac{w}{\|\mathbf{w}\|\,\|\mathbf{a}\|} + \frac{(\mathbf{w}\cdot\mathbf{a})\mathbf{a}}{\|\mathbf{a}\|^3\,\|\mathbf{w}\|} && (3.3)
\end{aligned}
$$

Because (3.3) contains the cosine function $\frac{(\mathbf{w}\cdot\mathbf{a})}{\|\mathbf{a}\|\|\mathbf{w}\|}$ hidden within it, we can cap its contribution to increasing the gradient. In fact, it can never multiplicatively increase the operand that contains it as values of cosine are between -1 and 1. If we set it to 1 (-1 would only change the sign, which does not matter when it comes to the magnitude of the gradient) for that part of the equation as a worst-case scenario (where "worst-case" refers to the least favorable conditions for the defense), we are left with

$$\frac{\mathbf{w}}{\|\mathbf{w}\|\,\|\mathbf{a}\|} + \frac{\mathbf{a}}{\|\mathbf{a}\|^2} = \frac{\mathbf{w}}{\|\mathbf{w}\|\,\|\mathbf{a}\|} + \frac{\mathbf{a}}{\mathbf{a}^T\mathbf{a}} \tag{3.4}$$

Adding an element with value q to $\mathbf{a}$ (which is of length l) results in the second term of the derivative's addition turning from $\frac{(\mathbf{a}\cdot 2)}{\left[\sum_{i=0}^{l} a_i^2\right]}$ to $\frac{(\mathbf{a}\cdot 2)}{\left[\sum_{i=0}^{l} a_i^2\right]+q^2}$. Notice that the derivative with respect to a_i scales with q^2, so it is not an ideal linear scaling that spreads the gradient across the vector proportionally. However, this may pass the test of the dot product's gradient contributions described in the beginning of this section, the only issue possibly being that the quadratic scaling imposed by q may cause an inversion of the problem, where large vectors become less affected by perturbations

than short ones, at least for added qs that are greater than or equal to one (the "equal to one" case would give us the aforementioned perfect scaling).

In regard to the same scenario with respect to the first term, we see that that term's contribution scales sub-linearly when adding that same q to $\mathbf{a}$, as the term of the denominator is the norm of $\mathbf{a}$, not its squared norm. This *reduces* the dot product issue, but does not completely mitigate it. However (though we have not analytically shown it), when in combination with the second term's scaling, it may be that they cancel out somewhat to give us a more linear scaling overall. The conjectures about this second term of the addition depend on $\mathbf{w}$'s requisite increase in the number of elements not throwing a wrench in the works; however, $\mathbf{w}$ is static at test time, so this may not matter.

It is important to note that, in the end result of (3.4), the term on the right is divided by the squared norm of $\mathbf{a}$, our input. If the elements of $\mathbf{a}$ become too small, you start to see that fraction grow in magnitude (because $a^T a \to 0$, while each individual element of a that it is divided by decreases linearly). In the case of MNIST[LCB], this becomes a serious issue if the images are fed directly into the network; this is due to the fact that the inverted version of MNIST (as used in [Sze+14b]) has many locations where the input $\mathbf{a}_0$ into layer 0 of an image (a) completely, or (b) almost completely consists of zero elements. For case (a), the derivative is even undefined. There is a simple solution to this, however. If we put a constant value c into the denominator of the cosine,

the derivative becomes

$$\frac{\partial}{\partial a}\frac{\mathbf{w}\cdot\mathbf{a}}{\|\mathbf{w}\|\|\mathbf{a}\|}$$

$$= \frac{\|\mathbf{w}\|\|\mathbf{a}\|\frac{\partial}{\partial\mathbf{a}}\mathbf{w}\cdot\mathbf{a} - \mathbf{w}\cdot\mathbf{a}\frac{\partial}{\partial\mathbf{a}}\left(\|\mathbf{w}\|\|\mathbf{a}\|+c\right)}{\left(\|\mathbf{w}\|\|\mathbf{a}\|+c\right)^2} \qquad \text{according to the quotient rule}$$

$$= \frac{\|\mathbf{w}\|\|\mathbf{a}\|\mathbf{w} - \mathbf{w}\cdot\mathbf{a}\|\mathbf{w}\|\mathbf{a}\cdot 2}{\left(\|\mathbf{w}\|\|\mathbf{a}\|+c\right)^2\|a\|} \qquad \text{[PP12, eqn. 69] and [PP12, eqn. 131] again}$$

$$= \frac{\|\mathbf{w}\|\|\mathbf{a}\|\mathbf{w} - \mathbf{w}\cdot\mathbf{a}\|\mathbf{w}\|\mathbf{a}\cdot 2}{\left(\|\mathbf{w}\|\|\mathbf{a}\|^2+2c\|\mathbf{w}\|\|\mathbf{a}\|+c^2\right)\|a\|}$$

$$= \frac{\|\mathbf{w}\|\|\mathbf{a}\|\mathbf{w} - \mathbf{w}\cdot\mathbf{a}\|\mathbf{w}\|\mathbf{a}\cdot 2}{\left(\|\mathbf{w}\|^2\|\mathbf{a}\|^3+2c\|\mathbf{w}\|\|\mathbf{a}\|^2+c^2\right)}$$

Notice that, as the norm of $\mathbf{a}$ increases, this modification starts to look more like what you would get from cosine, and the derivative does the same with respect to cosine's derivative (for the latter case, $\|\mathbf{w}\|^2\|\mathbf{a}\|^3$ drastically outpaces both the $2c\|\mathbf{w}\|\|\mathbf{a}\|^2$ term, and c^2 stays constant). Therefore, we get the reduced derivative behavior of cosine for large norms. As the norm gets smaller, the first and second aforementioned terms go to 0, leaving us with the third constant term. When this occurs, the cosine transforms into a scaled dot product (with scaling factor $\frac{1}{c}$), and, as a result, the derivative (by [PP12][eqn. 69]) becomes $\frac{1}{c^2}\mathbf{w}$, thus avoiding the trend of the derivative going to infinity. A downside is that $\mathbf{a}$s that cause this dot product-like behavior do not get the same protections as what is afforded by cosine. Further, at the time of writing, it is not clear to the author if and when a certain value of c is needed to prevent adversarial examples from exploiting even a little boost to the derivative by a shrinking norm.

3.3 Evaluation

3.3.1 Evaluation Procedures

One of the go-to [Sze+14b; KKG18] datasets when it comes to testing defenses is MNIST [LCB]. Modern papers [KKG18; Tra+20] use ImageNet [Rus+15] as a training/assessment database. How-

ever we did not get to considering ImageNet due to deadlines. So, we kept our evaluation to MNIST[3]. This dataset involves 70,000 images, 60,000 of which are meant for training, and 10,000 for testing. We chose 4,200 of the 60k as validation images (used for early stopping), and 55,800 used for the optimization procedure. The validation dataset was not subject to an adversary. Moving on, each image is 28 pixels wide and tall, with a single channel that represents grayscale data. The maximum value of a pixel is 255 (with the minimum at 0); however, unlike what is common in imagery, 255 represents black, and decreasing values cause increasing whiteness. This is an important point: [Sze+14b], for example, understandably, has 0 and 255 be the black and white, respectively. In order to be consistent with testing, we followed their assumption and did this same. Following [Sze+14b], in which they claim to have trained their networks for perfect *training set* classification, we unsuccessfully tried to do the same, but got close[4].

We chose the "FC-100-100-10" network from [Sze+14b] as our test network. However, it was necessary in some cases to modify the network as Pairwise Difference and Angular are not standard neurons. Further, in the latter case, we didn't use an activation at all; this is due to cosine having curvature and putting out values between -1 and 1 (note that these values are actually scaled depending on the value of its denominator constant). For Pairwise Difference, we used sigmoid as the final output of the layer. Because FC-100-100-10 is fully-connected, this meant that both Pairwise Difference and Angular took as input the output of the last layer or the image itself. Finally, the Pairwise Difference (which was only used for the first layer due to its pixel-oriented nature) and Angular-based networks used Batch Renormalization [Iof17], and we trained a normal FC-100-100-10 network with Batch Renormalization for the purposes of an ablation study (as we had found in the past that Batch Normalization [IS15] acted as a defense itself). In fact, the results for the network outside of ablation are interesting from an adversarial standpoint, so we include

[3]MNIST was suggested by Ali Varamesh as well; he was inspired by [Sze+14b] and [GSS15]

[4]We had seen a reference that did not achieve this either, implying that we are not alone and this might be normal, but have lost track of the reference

them for that reason as well.

Our complete implementation, found at [Cut][Friendly], used Chainer [Tok+19; Pyt; Har+20; Oku+17]. The training procedures outside of what has been described were very straightforward. We used momentum-based Stochastic Gradient Descent, and learning rates varied to fit the model more appropriately; momentum was kept the same for all models, as well as the batch size (20). The main reason for such a small number is that Pairwise Difference can use up a lot of memory, and we thought it fair that all networks use the same batch size for proper comparison. Early stopping was also employed, but none of our networks ever hit its 20 epoch limit (based on validation set performance) during the 100 epoch hard "deadline", so to speak. Prediction loss was based on mean squared error, and it was formulated as

$$\frac{1}{s}\sum_{i=1}^{s}(\mathbf{c}_i - \mathbf{o}_i)^T(\mathbf{c}_i - \mathbf{o}_i)$$

where s is the number of samples in the batch, $\mathbf{c}$ is the correct confidence vector (a.k.a. one-hot), and $\mathbf{o}$ is the outputs of the network for sample i. In the case of the network whose derivative is penalized, that penalty is given a weight of 0.5, while the mean squared error term has weight 1.0. See table 3.1 for the full list of hyperparameter settings that vary between networks. Note that we tried only to follow through mostly with point 2 and partially with point 1 in 3.1.4; time prevented us from exhausing all three points. Specific to point 1, we do not achieve the properties "any compelling threat model should at the very least grant knowledge of the model architecture, training algorithm, and allow query access" [ACW18][§ 6.1] and "[it] is not meaningful to restrict the computational power of an adversary artificially (e.g., to fewer than several thousand attack iterations)" [ACW18][§ 6.1]. As is stated in [ACW18], it is important to conduct tests using a separate network for generating adversarial examples due to "gradient masking" (as a reminder, this is an effect where the trained network can handle attacks that use the network's gradient, while not being able to handle other kinds of attacks; this author only knows of gradient-based attacks which use a different network, but there may be something he is missing). As a result, they

	S	RS	A	GMR	GMRES	PD
Batch Renormalization	no	yes	yes	no	no	yes
Activation function	sigmoid	sigmoid	N/A	sigmoid	Elastic Sigmoid	sigmoid
Learning rate	0.002	0.002	0.0002	0.0009	0.002	0.002

Table 3.1: "S" stands for "Standard", "RS" for "Renormalized Standard", "A" for "Angular", "GMR" for GMR, "GMRES" for the addition of Elastic Sigmoid to GMR, and "PD" for "Pairwise Difference"

	S	RS	A	GMR	GMRES	PD
White-box (FGSM)	46.16%	43.30%	0.53%	37.67%	36.98%	32.20%
Black-box (FGSM)	45.28%	45.27%	56.35%	43.24%	44.70%	49.87

Table 3.2: Performance of each network when it comes to black-box and white-box attacks. Table 3.1's caption has the meaning of the top-row acronyms.

recommend black-box attacks, where attacks are performed not using any internal values (such as the gradient) from the network being tested. As a result, we also show black-box results where we train another[5] original, non-protected network provides the adversarial examples to each of the defended networks. The results on the MNIST test set can be found in table 3.2. We followed [Mad+19] and used the $[0, 255]$ equivalent (77) of the 0.3 (out of 1.0) L_∞ noise that they chose. We use L_∞ as it is used throughout the literature ([GSS15; Mad+19; Tra+20], etc) and is the easiest to program for. To disclose discrepancies in performance under normal circumstances, we present table 3.3. All testing was performed *not* using the values Batch Renormalization learned (which turns it into Batch Normalization [IS15], according to [Iof17][§ 3]) as we had very strange (at least from the perspective of our knowledge of Batch Renormalization) performance issues otherwise; this issue may be the author's fault, but we did not get to resolving it.

[5] [Mad+19; Sze+14b] are the inspirations for "another"

S	RS	A	GMR	GMRES	PD
95.86%	97.80%	85.40%	96.09%	96.72%	95.91%

Table 3.3: Same layout as table 3.2, but the networks were not attacked.

3.3.2 Outcomes

The results turned out to be poor. It is questionable whether or not we can make any claims whatsoever. Angular's adversarial performance was just bad (even relative to a normal network without defenses) when up against a white-box attack, a surprising find. It did do pretty well for the black-box equivalent, but, again, its performance in table 3.3 makes this network nearly useless considering that most scenarios in real life will not be adversarial. Further, such comparisons are not fair because Angular is not in the same performance "state" as the other networks; this is why [Sze+14b] tries to keep performance across networks the same. If we were to actually get the angular network's accuracy up enough, it could be that the black-box benefits would disappear. It also turned out that there was no point to ablation because there were no benefits of using Batch (Re)normalization. Speculation about the results can be found in 3.3.3.

3.3.3 Analysis

The performance of an Angular-based network is simultaneously surprising and possibly easy to explain. The first rationale could be that we did not pick an adequate constant to nullify the aforementioned issue of many of zero-length vectors. None of these vectors actually ended up a zero because each of them contained the whole image (a vector with all-zero values would contain no number pixels), but the constant may cause issues regardless if not chosen properly. However, a more likely scenario is that the scaling of the gradient with the addition of more elements is not as good as hoped. As stated previously, ideally the gradient of each element in the input vector of an Angular neuron contributes a normalized amount such that all partial derivatives in the vector

add up to, roughly speaking, the same value as when using a small patch as input. Such a neuron would not be too dissimilar from Angular, but coming up with the formulation that we would want probably entails integrating from the derivative that we want. We may even be able to drop normalizing the weight vector by dividing it by $\|\mathbf{w}\|$ entirely, seeing as it is held constant at time of attack. Though hardly fleshed-out, we would need to integrate something similar to $\frac{\mathbf{a}}{\|\mathbf{a}\|_1}$ (this does not involve the weight vector, so this integration is more complicated than what is shown).

As far as Pairwise Difference, the most likely explanation is that most pairs involve a comparison between two zero-value pixels, which puts each subtraction squarely in the middle of the sigmoid used for comparison. As a result, even FGSM using a max-norm of just 10 has a reduced 34.50% accuracy. It is possible that treating each comparison as a miniature neuron with two weights (or even a bias) could be a fix for this issue because, as previously stated, this would still attempt to address the issue of large vectors in dot products.

Renormalization appears to have been worse for the original network, which actually checks out. This is due to the fact that [IS15] states that Batch Normalization [IS15] (Renormalization's predecessor) aims to keep the input to the activation function centered near, and tight around, the origin (at least in the case of the sigmoid function) which is where the strongest gradients lie. While this is a blessing for optimization during training, it's also a blessing for any attacker who uses gradients in their attacks. It is possible that one conclusion that should be drawn from this is that a Batch Renormalization-trained network in Batch Normalization mode appears to be detrimental from a defensive standpoint.

Gradient Magnitude Reduction's lack of usefulness is concerning. The theory behind GMR, at least intuitively, makes sense, and this may just come down to using the wrong weight in the loss term involving the gradient w.r.t. the image. However, an important point to raise is that GMR does not make any guarantees about reducing the gradient in any location other than at the point at which the gradient is calculated. It may help to penalize even the second derivative (as it has

a hand in determining the first derivative in other locations), but the Universal Approximation Theorem[HSW89] makes it possible that even the second derivative may not be enough seeing as there may be a valid third derivative, fourth derivative, etc. It is relevant to note that adversarial training side steps this issue by essentially reducing the gradient at points far (relatively speaking) from the normal datapoint, forcing (according to the evidence) all derivatives (not just the first) to be close to zero. Elastic Sigmoid paired with GMR was ineffective as well, for the same reasons.

Clearly, we have good reason to look further into all the defenses proposed; future work will need to be responsible for doing so. We also need to consider Projected Gradient Descent [Mad+19] as an attack, as recommended by [ACW18] because of its effectiveness.

Chapter 4

Conclusion

Representation is a very important part of neural networks (and machine learning in general). We have put an emphasis on subitization as well as adversarial examples in order to discuss this topic, and shown what appears to be success in representation learning via learning to count. More complex situations need to be explored, as well as changes to and full implementation of our detection methods to really make it work, and what we have seen with our experiments hints at this truly being possible. We did not demonstrate a network with full subitization capabilities, but modifications to the training and/or network may make this a reality.

As for adversarial examples, we did not have success, but we did lay out substantial justification for the defenses that we proposed; hopefully, this analysis leads us to (a) related method(s) that corrects for any analytical flaws in what we have proposed. Specifically, we emphasized structure over additional loss terms, with the exception of GMR. However, even a modified form of GMR, in the future, might benefit from structural changes, such as when any of these modifications are used in conjunction with methods such as Elastic Sigmoid. While papers like *Adversarial Logit Pairing* [KKG18] appear to be very close to a cure, the space of attacks seems boundless, and this author believes that mathematical proofs are required in order to fully address this issue. In the author's opinion, proving that iterative optimizers and special loss terms really achieve ideal networks is difficult due to the dynamic nature of these methods and the infinite set of points over which they are optimizing. Seeing as methods like Adversarial Logit Pairing and adversarial training are defenses that fit within this definition, the author is somewhat skeptical about their true effectiveness. This is because adversarial examples like what can be found in [Eyk+18] focus on fooling networks relative to all human perception, and noise bounded by L_∞ or similar metrics clearly does not model this. Structural changes appear easier to prove (from an adversarial example

view), so they may make the most sense to pursue further. In the end, we look into this topic because the adversarial example community inherently focuses nearly completely on representation.

In summation, topic of representation needs substantially more resources dedicated to it. For the author, this thesis is merely a starting point, and we plan to build upon this work. We need to solidify the fundamentals before we can truly unleash deep learning.

Bibliography

[ACF76] J. Atkinson, F.W. Campbell, and M.R. Francis. "The magic number 4±0: a new look at visual numerosity judgments." In: *Perception* 5.3 (1976), pp. 327–334. ISSN: 03010066. URL: https://proxyiub.uits.iu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-0017099920&site=eds-live&scope=site.

[ACW18] Anish Athalye, Nicholas Carlini, and David Wagner. *Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples*. 2018. arXiv: 1802.00420 [cs.LG].

[ALZ16] C. Arteta, V. Lempitsky, and A. Zisserman. "Counting in the Wild". In: *European Conference on Computer Vision*. 2016.

[Ama] Amazon Mechanical Turk. *Amazon Mechanical Turk*. URL: https://www.mturk.com.

[Ath+18] Anish Athalye et al. *Synthesizing Robust Adversarial Examples*. 2018. arXiv: 1707.07397 [cs.CV].

[BJ01] Y.Y. Boykov and M.-P. Jolly. "Interactive graph cuts for optimal boundary amp; region segmentation of objects in N-D images". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 1. 2001, 105–112 vol.1. DOI: 10.1109/ICCV.2001.937505.

[Cm] Alex Clark and more. *Pillow*. URL: https://github.com/python-pillow/Pillow.

[Cut] Benjamin Cutilli. *CountingPlusFriendly*. URL: https://github.com/benvcutilli/CountingPlusFriendly.

[Dic] Dictionary.com. *Subitize Definition & Meaning — Dictionary.com*. URL: https://www.dictionary.com/browse/subitize (visited on 11/15/2021).

[Dug+01] Charles Dugas et al. “Incorporating Second-Order Functional Knowledge for Better Option Pricing”. In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper/2000/file/44968aece94f667e4095002d140b5896-Paper.pdf.

[Eyk+18] Kevin Eykholt et al. *Robust Physical-World Attacks on Deep Learning Models*. 2018. arXiv: 1707.08945 [cs.CR].

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Wikipedia (https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) claims this reference as the one for ReLU. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[Gil+–] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects*. toblerity.org, 2007–. URL: https://github.com/Toblerity/Shapely.

[GL21] Shuyue Guan and Murray Loew. *Understanding the Ability of Deep Neural Networks to Count Connected Components in Images*. 2021. arXiv: 2101.01386 [cs.CV].

[Goo+13] Ian J. Goodfellow et al. *Maxout Networks*. 2013. arXiv: 1302.4389 [stat.ML].

[GR15] Shixiang Gu and Luca Rigazio. *Towards Deep Neural Network Architectures Robust to Adversarial Examples*. 2015. arXiv: 1412.5068 [cs.LG].

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[Har+20] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[Iof17] Sergey Ioffe. *Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models*. 2017. arXiv: 1702.03275 [cs.LG].

[IS15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[Kau+49] E. L. Kaufman et al. "The Discrimination of Visual Number". In: *The American Journal of Psychology* 62.4 (1949), pp. 498–525. ISSN: 00029556. URL: http://www.jstor.org/stable/1418556.

[KKG18] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. *Adversarial Logit Pairing*. 2018. arXiv: 1803.06373 [cs.LG].

[Kri09] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[LCB] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *The MNIST Database of Handwritten Digits*. The MNIST Database of Handwritten Digits.

[LKX] Fei-Fei Li, Ranjay Krishna, and Danfei Xu. *CS231n Convolutional Neural Networks for Visual Recognition*. I assume that authors, from "Instructors" of http://cs231n.stanford.edu, are listed in order of importance, so we copied that order for this reference. URL: https://cs231n.github.io/optimization-2/ (visited on 11/10/2021).

[LN89] Dong C. Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical Programming* 45 (Aug. 1989), pp. 503–528. DOI: 10.1007/BF01589116.

[LZ] Victor Lempitsky and Andrew Zisserman. “Learning to Count Objects in Images”. In: Neural Information Processing Systems (2011).

[Mad+19] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML].

[Mic] Microsoft Corporation. *Paint 3D*. Comes with Windows 10. URL: https://www.microsoft.com/en-us/p/paint-3d/9nblggh5fv99.

[MS82] George Mandler and Billie Jo Shebo. “Subitizing: An analysis of its component processes. Journal of Experimental Psychology: General, 111, 1-22.” In: (1982). URL: https://proxyiub.uits.iu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edssch&AN=edssch.oai%3aescholarship.org%2fark%3a%2f13030%2fqt9fn27772&site=eds-live&scope=site.

[Mun+16] T. Nathan Mundhenk et al. “A Large Contextual Dataset for Classification, Detection and Counting of Cars with Deep Learning”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 785–800. ISBN: 978-3-319-46487-9.

[Ner14] Nerdwriter1. *Counting, Explained*. July 28, 2014. URL: https://www.youtube.com/watch?v=qRMP6rCT_bs.

[Net+11] Yuval Netzer et al. *Reading Digits in Natural Images with Unsupervised Feature Learning*. Included bibliographic information and its best approximation to BibLaTeX entries is from http://ufldl.stanford.edu/housenumbers/ (That URL’s use is also

requested by that site). 2011. URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[Oku+17] Ryosuke Okuta et al. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations". In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf.

[Pap+17] Nicolas Papernot et al. "Practical Black-Box Attacks against Machine Learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. Abu Dhabi, United Arab Emirates: Association for Computing Machinery, 2017, pp. 506–519. ISBN: 9781450349444. DOI: 10.1145/3052973.3053009. URL: https://doi.org/10.1145/3052973.3053009.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[PP12] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. Version 20121115; https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf was where it was originally gotten from. Nov. 2012. URL: http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html.

[Pyt] Python Software Foundation. *Python*. Version 3. URL: python.org.

[Red+16] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[Ree21] Michael Reeves. *Teaching a Robot Dog to Pee Beer*. As a warning, given the formality of this thesis, one should be aware of the crudeness of this video. Apr. 9, 2021. URL: https://youtu.be/tqsy9Wtr1qE.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323 (Oct. 9, 1986), pp. 533–536. DOI: 10.1038/323533a0.

[rMD18] radouxju, Martin Thoma, and Devdatta Tengshe. *algorithm - Finding if two polygons Intersect in Python? - Geographic Information Systems Stack Exchange*. Sept. 14, 2018. URL: https://gis.stackexchange.com/questions/90055/finding-if-two-polygons-intersect-in-python.

[Rus+15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. 2014. arXiv: 1312.6034 [cs.CV].

[Sze+14a] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[Sze+14b] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199 [cs.CV].

[Sze+15] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[Tok+19] Seiya Tokui et al. “Chainer: A Deep Learning Framework for Accelerating the Research Cycle”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, pp. 2002–2011.

[Tra+17] Florian Tramèr et al. *The Space of Transferable Adversarial Examples*. 2017. arXiv: 1704.03453 [stat.ML].

[Tra+20] Florian Tramèr et al. *Ensemble Adversarial Training: Attacks and Defenses*. 2020. arXiv: 1705.07204 [stat.ML].

[var] various. *Distance matrix - Wikipedia*. Not sure which version or when it was read; likely late 2017 for the former and the same time in 2018 for the latter. Via Googling. URL: https://en.wikipedia.org/wiki/Distance_matrix.

[Yos+15] Jason Yosinski et al. *Understanding Neural Networks Through Deep Visualization*. 2015. arXiv: 1506.06579 [cs.CV].

[Zha+15] Jianming Zhang et al. “Salient Object Subitizing”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.