THESIS

COUNTING WITH CONVOLUTIONAL NEURAL NETWORKS

Submitted by

Viraj Shastri

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2021

Master's Committee:

Advisor: J. Ross Beveridge

Nathaniel Blanchard
Christopher Peterson

ABSTRACT

COUNTING WITH CONVOLUTIONAL NEURAL NETWORKS

In this work, we tackle the question: Can neural networks count? More precisely, given an input image with a certain number of objects, can a neural network tell how many are there? To study this, we create a synthetic dataset consisting of black and white images with variable numbers of white triangles on a black background, oriented right-side up, down, left or right. We train a network to count the right-side up triangles; specifically, we see this as a closed-set classification problem where the class is the number of right-side up triangles in the image. These evaluations show that our networks, even in their simplest designs, are able to count a particular object in an image with a very small epsilon of approximation. We conclude that the neural networks are enforced with more complex learning capabilities than given credit for.

ACKNOWLEDGEMENTS

I would like to thank my advisors Dr. Ross Beveridge and Dr. Nate Blanchard for the guidance, patience and encouragement through this comically arduous year of online classes, and a pandemic. A special thanks to Dhruva Patil for the constant help in researching for and writing this thesis.

Additionally, I would like to thank my fellow graduate students for listening to, and critiquing this work to help me present it in the best way possible. I would like to extend my gratitude to my parents and my family for always having my back and believing in me. A special thanks to Elliot Forney for creating this LaTeXtemplate that vastly sped up the writing portion of this thesis. Lastly, I would like to thank the Department of Computer Science at Colorado State University for providing me the opportunity to work with such wonderful people and giving me a delightful research experience this early on in my career.

DEDICATION

I would like to dedicate this thesis to my parents, Smita and Vishwanath Shastri.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This work answers the basic question, *"Can everyday neural networks count?"*. Before we approach this question, we must first define what the term "counting" means to an artificial neural network. Consider an example where there are "61" objects in a given input image and when that image is passed through a *convolutional neural network*, the network tells us that there are "64" objects in that image, do we consider that as an acceptable answer? If so, can one say that the network is counting? Or is it just estimating the number of objects in the given image by accepting any value that lies within a certain range of the actual count. This thesis tries to answer this question by focusing the task of a neural network solely on counting and for doing so, introduces a new synthetic dataset entitled *"The Triangles Counting Dataset."*

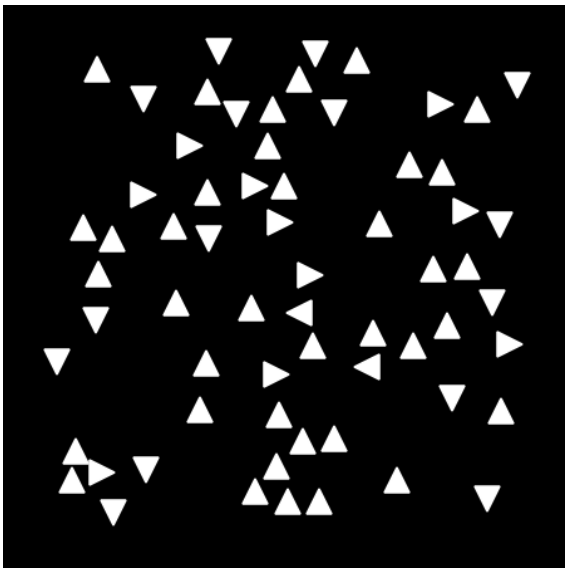
Most work in counting with neural network comes under the domain of crowd counting. This essentially is an open-ended problem where an image may contain from zero to any number of objects, in this case people. Researchers often focus on creating a network that can most accurately estimate the number of people in that image instead of trying to figure out the exact count. The reason for this is because these images, in addition to people, contain other objects and also widely differ from each other due to differences in the crowd density, object sizes, perspective, image sizes, and color as shown in Figure 1.1c and Figure 1.1d. Thus, a synthetic dataset was needed that could provide a controlled environment to study what a neural network learns when it's tasked with counting objects in an image.

The Triangles Counting Dataset achieves this by limiting the scope of the information that can be inferred from an image. This dataset changes the question from an open-ended set of "how many objects are there in an image?" to a closed-set problem of "what happens when a neural network tries to count N number of objects in an image?" To that end, the Triangles dataset adopts a closed-set classification approach where an image will contain a maximum of N objects as showcased in Figure 1.1a and Figure 1.1b with 37 and 47 objects to be counted respectively. The value of N

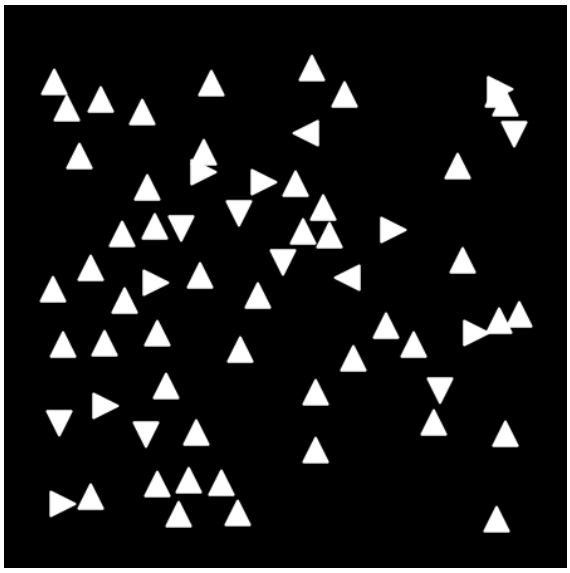
chosen is 64 as it gives a good balance of testing the extent to which a network can count and it is low enough to be used for a simple classification network. Since, the main objective is counting and not recognition, we use simple objects that the network can easily recognize, which in this case are triangles. These triangles are oriented in four directions; up, down, left and right. Section 3.3 explains in detail the exact structure of the dataset along with why it uses triangles instead of other shapes.

A very simple solution could be manufactured for the triangles problem. Such a solution could contain, for example, a series of filters that are constructed such that they can isolate each of the individual shapes present in the image. This would not only be faster, but also computationally less expensive than running a neural network. But, as mentioned before, this thesis is more about understanding a neural network as it tries to count. And, as the latter sections explore the outputs of the neural network, it does just that, that is, it creates a series of filters to identify the shapes. But is does so in a very interesting way and the most important part is that it does so without any additional information. In a way, this approach lies in the middle of a supervised and a reinforcement type approach.

The rest of this thesis is organized as follows. Chapter 2 discusses related work in counting using neural networks along with specific terms and concepts used. Chapter 3 describes the neural network architecture, the experimental setup and a brief overview of the final version of Triangles dataset. The iterative process of updating the dataset and the experiments conducted during each iteration are then elaborated in Chapter 4 followed by the conclusion and a brief discussion about the future work related to this research in Chapter 5.



(a) Count–37, Occlusion–0%.



(b) Count–47, Occlusion–20%.

(c) Shanghaitech Part A.

(d) Shanghaitech Part B.

Figure 1.1: Sample images from the Triangles Counting Dataset and Shanghaitech dataset. (a) No occlusion between triangles. (b) Up to 20% occlusion between triangles. (c) Sample from Shanghaitech Part A. (d) Sample from Shanghaitech Part B.

Chapter 2

Background

This chapter details related work in the domain of counting and some background information necessary for understanding the empirical study performed in Chapter 4. One of the most popular domain in counting with neural networks is crowd counting. Figure 1.1 showcases two images from the Shanghaitech dataset for crowd counting. Section 2.1 illustrates this and other popular crowd counting datasets as well as current techniques used for it.

2.1 Crowd Counting

Crowd counting, unlike the name suggests, is actually the task of *estimating* rather than *counting* the number of people in a given image. The image can contain from zero to any number of people depending on how dense or sparse the crowd is.

2.1.1 Traditional Detection based Approach

Traditional machine vision to perform crowd counting typically uses some combination of computer vision techniques like object detection, image segmentation, feature detection, etc. Viola *et al.* in [1] used image intensity information integrated with motion information trained over two consecutive frames of a video for pedestrian detection. The dataset used was a custom dataset of video sequences of street scenes with pedestrians annotated in a bounding box. Leibe *et al.* in [2] had the main goal to detect pedestrians, that was achieved by integrating local and global feature cues via a top-down segmentation. In the process, it also counted them as the detection was done by drawing bounding boxes around people. Here, the authors also created a special dataset for training consisting of images of a person with different outfits and accessories walking in front of a camera in front of two different backgrounds.

Both these approaches try to draw a bounding box around each person and then count the number of bounding boxes to get the count values. This is a strong contrast to the work done

in this thesis where neural networks were used to get a direct count of the objects in an image. Although, this is not a fair comparison as the two approaches are quite old and were published a while before CNNs became immensely popular with machine vision tasks.

2.1.2 Density Map Approach

Many modern approaches try to predict the density of the crowd with deep neural networks and later integrate it get a count estimate. Density prediction is done by generating *density maps* of the sample images to be used as the ground truth. The density map is an image generated by convolving a Gaussian kernel on a function on the coordinates of peoples' head positions. In a nutshell, the network is trained on the sample images to predict this density map.

Early work with using density map approach to count objects in images includes Lempitsky *et al.* [3] where SIFT detectors were used to generate a density map. The experiments were conducted on Lehmussola *et al*. [4] and Chan *et al.* [5] datasets where the images are annotated with the coordinates of objects.

Modern approaches make the use of neural networks based on CNNs to figure out the density function. Zhang *et al.* [6] first introduced a CNN based architecture to perform crowd counting using the density map approach. Zhang *et al.* [7] introduced along with another novel CNN based neural network architecture to predict the *density map* of an image, the Shanghaitech Dataset, a large scale dataset for crowd counting. The architecture used is a multi-column CNN based neural network where each *column* contains differently sized filters that identify heads of different sizes. This is necessary as heads are different sizes due to perspective distortion. Zhang *et al.* [8] took inspiration from this and modified AlexNet [9] to make a density map estimation. Chen *et al.* [10] improves upon this architecture by adding attention through a confidence module and fusing the confidence information with the density information to reduce errors when neural networks mistakenly detect other objects like dense shrubberies as human heads.

Arruda *et al.* [11] presents alternate applications for the counting algorithm introduced in the paper, although the underlying approach is the same. This approach was evaluated against a tree counting dataset and a more popular car counting benchmark introduced in [12].

An important note to make here is that these density map estimating methods require additional annotated information like the coordinate position along with the image. Unlike this work, where a cross-entropy loss is calculated by comparing the target label with the output label, these approaches use the density map to calculate the loss and not the count value. The labels used in this work essentially are the count values, although, for the neural network, they are nothing but labels.

2.1.3 Classification Approach

Häni *et al.* [13] uses a classification based approach similar to the work presented in this thesis. The authors modified AlexNet to output the count the objects, in this case, apples in a cluster in orchards in classes of zero to six count values. [14] and its follow-up [15] by Xiong *et al.* introduces a closed-set classification based approach for crowd counting, where the class represented intervals that the count of an image patch could belong to. These local counts were then later added in a *divide and conquer* fashion to obtain the final count.

The approach taken in this work is most similar to these two bodies of work and takes some inspiration from Xiong *et al.* [14]. A distinction is that we define the classes as the absolute count values of the entire image unlike in [14], that defined the classes as an interval of count values for partial sections of an image.

2.2 Visual Learning and Reasoning

The counting problem being studied in this work is a small example of a much larger class of problems. These problems get to the heart of whether or not modern neural networks learn to reason about what they see in anything approaching the way humans would. And further, of greatest practical importance, whether they make mistakes that people would never make.

A fairly recent paper in this general area includes Not-So-CLEVR ([16]). Here, the authors present the argument that modern CNN based feed forward deep learning networks are approaching human-level performance in tasks like object detection and classification. Even with this, these networks fail to learn visual relations. The authors explore whether networks can learn to interpret abstract information when tasked with learning the relations between objects in an image.

The study introduces a modified version of the SVRT Dataset introduced by Stabinger *et al.* in [17] that has synthetic images generated using framework presented in [18] to test whether networks learn abstract relations between objects in images. This modified version, called PSVRT, has images in two categories; same-different where the classes are whether the objects in an image are same or different sans translation, and spatial-relations where the classes indicate whether the objects are laid out vertically or horizontally with respect to each other in the image. The modifications were made to conduct strictly controlled experiments where each aspects of an image like the number and size of objects, image resolution was controlled per experiment. The goal of the experiment was not whether the network can predict the label of an unseen image, but rather whether the network can learn to identify such abstract relations.

The experiments include testing with deep and wide CNN based feed forward networks, relational networks, and Siamese networks. The CNN based networks, which seemed to perform well when tested with the SVRT dataset, failed to learn on the more controlled PSVRT dataset in the same-different category. The same networks, however, learned in the spatial-relations category. The authors theorized this to the fact that the networks, in the case of same-different category, instead of learning the relations between the objects, simply learned a collection of templates covering certain distributions over the image.

Another very interesting line of work involves what is called the Raven Dataset presented in [19] by Zhang *et al.* The synthetic images in this dataset are based on Raven's Progressive Matrices, that are designed to test the abstract and structural reasoning ability of neural networks. The images consists of simple gray-scale objects arranged in specific rule-bound structures. These objects are designed to be inherently light in recognition. The structure is what makes it challenging

in visual reasoning. The task here is to predict the next (9^{th}) image in a series of 8 given images. These are grouped together in sequences of three images, where each group follows the same rules for generating the next image.

2.2.1 A Comparison with this Thesis

The papers explored in Section 2.2 are heavily invested in studying the visual reasoning capabilities of neural networks. Visual reasoning, by and large, is a combination of various smaller tasks for example, object detection, segmentation, counting, etc. Studying such networks is a difficult task as the scope is wide and encompasses multiple things that the network can learn including but not limited to the different type and number of objects, their positions in an image, their positions relative to each other, etc.

In the thesis that follows, the scope is precise and focused. In counting, the output is supposed to be one right answer, no more, no less. This is unlike the papers explored in Section 2.1 where the networks are assigned with estimating the count of objects. The scope gets widened here as the network has to deal with multiple variables like different object sizes, image sizes, perspective loss, etc. We, on the other hand, limit the variables that the network can learn to by using a synthetic dataset instead of "real-life" datasets like crowd-counting datasets. The synthetic images use simple objects like triangles that are easy on recognition. The size of all the objects in an image is kept the same along with the image size. In place of having the number of objects in a dataset having a ranging from 10s to 1000s (Shanghaitech Dataset [6]), the number of objects is limited to a certain number and a classification approach is used. The network's goal is to classify the image into a class corresponding to its object count.

Chapter 3

Methodology

The approach taken in this thesis is to generate a series of synthetic datasets that enable performing controlled experiments on different CNN based architectures. These experiments involve evaluating these networks with these carefully generated datasets with the goal of appropriately gauging the extent to which said model can count. The study empirically measures the degree up to which a neural network can successfully count the number of objects in an image.

"Success" in the context of counting depends on what is defined as *counting* as mentioned in Chapter 1. There is an important distinction between *approximately correct* and *correct*. This thesis does not concern with getting an approximate count as that task comes under the title "count estimation." Counting involves getting the correct answer and as such, the experiments performed in this work highlights what a neural network learns when tasked with counting. The experiments also explore near misses, or in other words, missing the exact count by a couple values, when counting.

The network used for the experiments remained unchanged for the most part with a few modifications later on in the trials that are detailed in Section 3.1. The synthetic dataset was iteratively altered based on the observations made when the network was evaluated with a certain version of the dataset. The observations were made by studying specific performance metrics generated during the evaluation process.

Section 3.1 first explains in brief the neural network architecture used in these experiments. Next, Section 3.2 details the experimental setup including the steps taken for data processing, training and validating the network, and an insight into the performance metrics used. Finally, Section 3.3 showcases the proposed Triangles dataset.

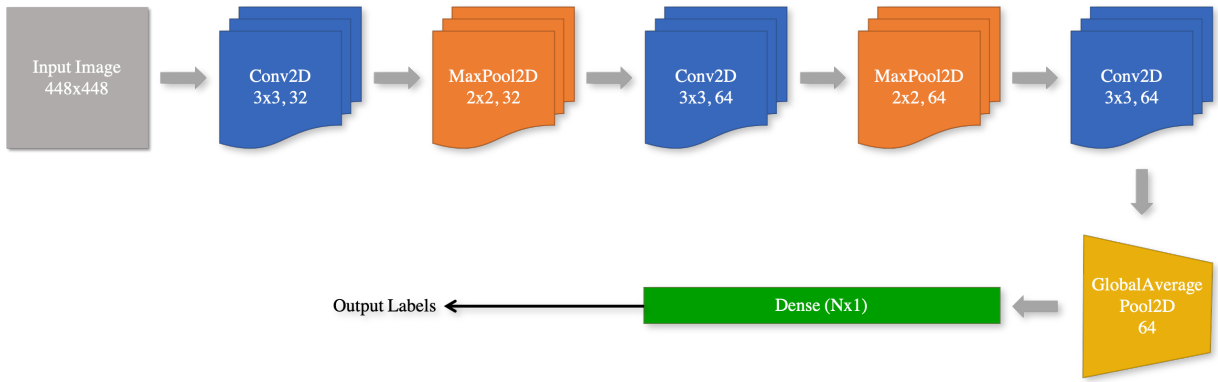


Figure 3.1: Baseline network architecture for the CNN based model

3.1 CNN Architecture

Figure 3.1 shows the architecture of the model that was used as a starting point for the experiments. The input to the network varied with the evolution of the dataset. In more precise terms, the input is an image that started with a resolution of 32x32 pixels with the first iteration of the dataset that increased up to 448x448 by the final iteration; more on this in Chapter 4. The input is in the form a NumPy array of the shape 448x448x1 for the final version of the synthetic dataset with previous versions having the input size that corresponded to the respective image resolution. The feature extractor consists of three convolutional layers with one max pooling layer between each pair. All three convolutional layers have a kernel size of 3x3 while the max pooling layers have a kernel size of 2x2. The first convolutional and max pooling layer have 32 filters while the rest have 64 filters each. TensorFlow default values were used for the filter stride.

The classification layer consists of a global average pooling layer that takes the average of each of the 64 output filters of the 3rd convolutional layer. It is followed by the final layer which is a simple dense layer with 65 units since an image can have zero to 64 right-side up triangles. Although, this was, like the input, different for the previous iterations of the dataset depending on the number of classes in those iterations as will be discussed further in Chapter 4. The minor modification made to the network in a later experiment consisted of swapping the global average pooling layer for a global max pooling layer to explore the differences in the evaluation results of the network.

3.2 Experimental Setup

The overall key step in these experiments consisted of generating a synthetic dataset where the number *and* type of objects in an image was strictly controlled. The network was evaluated on these images to generate specific performance metrics, for example *activation maps*. These performance metrics helped showcase what the network learned when trying to count the number of objects of a specific type in that image.

The experimental setup consisted of first generating the images for the dataset with varying number of objects to count. The dataset was then split into training, validation and test sets followed by training and validating the network on them. The test set was used to generate all the performance metrics detailed in Section 3.2.1. This section presents in detail the data preparation step, the training and validation of the network using TensorFlow and finally the performance metrics that are generated when testing the network.

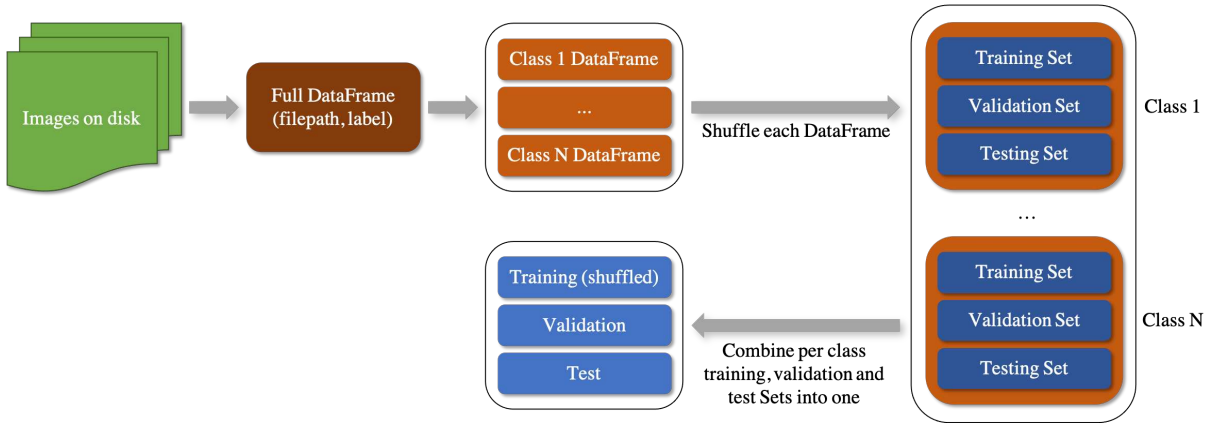


Figure 3.2: A step-by-step process for setting up the dataset for evaluating the network.

The file paths of the images were parsed to extract the class label of the images; both these values were stored as a tuple in a Pandas DataFrame. The next step consisted of splitting the DataFrame into three sets, namely, training, validation and testing. To make sure that the data was equally distributed in these sets, the DataFrame was first split based on the class labels where the file paths in each individual split belonged to a single class which were then individually shuffled.

Next, these splits were then individually divided into training, validation and test sets in a 60:10:30 ratio respectively. Lastly, the final training, validation and test sets were created by joining these individual splits. This process is showcased in Figure 3.2.

To feed the data to the network, TensorFlow-Keras flow generator was used. This allowed us to use the previously created DataFrames to automatically generate *input-label* tuples that can be fed to the network in batches if necessary. Internally, it uses the Pillow library to read the images from the file paths as single channel black and white images, which, for example, gave us a 448x448x1 NumPy array for each image of the current version. Finally, the images were normalized, again through the flow generator itself, by dividing the array by 255 to get floating point values between 0 and 1.

With this, the network was ready to be evaluated; the fit function of TensorFlow was used for training the network which used a training and a validation split during training itself. The validation split in this case is used by the fit function to generate the training loss and accuracy of the model after each epoch. The network was trained for up to 20 epochs with categorical cross-entropy as the loss function and Adam optimizer with an initial learning rate of 1e-3. A decay of 1e-1 was added that was applied to the learning rate if the training loss calculated after two consecutive epochs was less than 0.5, a phenomenon also known as plateauing.

3.2.1 Performance Metrics

The performance of the trained model was recorded in three ways; first, by recording the evaluation loss and accuracy calculated on the test set of a particular dataset after training the network on it, second, by generating a confusion matrix using the same test set, and third, by plotting the activation map of the feature extractor when the network was given a sample image from the test set.

The confusion matrix is a 65x65 matrix with the target labels on the y-axis and predicted labels on the x-axis. Although, the size of the matrix changed with each iteration of the dataset as each

version had different number of objects to count. For example, the 3x3 Grid had 10 output units, and the Fixed 3x3 dataset had four output units in the final dense layer.

Meanwhile, the activation map was generated for each filter of the first five layers of the network, i.e. the feature extraction layer, although, the focus was only on the activation map of the final convolution layer. This helped visualize what each filter *looked at* for a particular image. The terms "activation map" and "activations," both refer to the activation map of only the last convolutional layer henceforth in this thesis. This activation map contains the output of the 64 filters of the final convolutional layer arranged in a 4x16 grid layout with each position in the grid containing an image of the activation of that filter. The plots are titled with the ground truth label (T) and prediction (Y) of the sample image that was used to generate the map.

3.3 Datasets

All versions of the datasets tested with the network shared certain commonalities such as having only single channel images, i.e. black and white images, in the dataset, and having fixed number of objects to be counted in the images. The objects were white in color scattered on a solid black background. The number of objects to be counted in an image determined the class label for that image. Overall, the number of objects to be counted that were tested in the experiments ranged from a minimum of zero to a maximum of 64.

The datasets were generated iteratively with the conclusions drawn from observing the performance metrics of each version influencing the design of the next. Chapter 4 details the experiments performed on each version of the dataset. For reference, the iterations follow the order Dots, Circle-Diamond, 3x3 Grid, Fixed Grid, 2-Shapes, and finally 4-Shapes; these are the nicknames for each version for better understanding the results in the following chapters.

While the following chapter details the dataset, here's a quick overview of the construction of the dataset(s). To control what the network can infer from the image, the dimensions explored in the construction are the marker (object) type, placement, and count.

The marker type started with a simple white dot only a couple pixels wide in a 32x32 image. This was difficult for the network to recognize due to the relatively small resolution of the object as well as the image. An additional problem introduced by it was linear separability based on average gray levels of the image. The next was using solid white circles and diamonds approximately 22 pixels wide. In this version, the diamonds were added to reduce the linear separability but were found to be ineffective as getting the area of both these shapes to be the same turned out to be quite difficult. Finally, triangles were chosen to be the marker, again 22 pixels wide. The orientation of the triangles could be changed to introduce *different* objects without changing the area of both these objects. This eliminated the average gray level difference between the classes that indicated that the dataset was not linearly separable and the network now will have to identify the objects that it was supposed to count.

The placement of the object first was random when using the dot marker. This was possible as the object was relatively small. When shifted to circles and diamonds, the placement was fixed in a grid layout to ease the image generation process by avoiding occlusion due to the increased size of the object with respect to the image size. The same idea is used for the initial versions when using the triangle markers. Later, the placement was again randomized when the image size was increased (doubled).

Finally, the number of objects remained the same for the first half of the study at 9. In the grid layout, a 3x3 layout was used. It was then increased to 16, 25, and 36 in a 4x4, 5x5, and 6x6 layout respectively to test the extent of the networks counting ability. Keeping the image size the same, 36 objects made the image too dense and hence, the image size was doubled. This allowed the final version of the dataset to have 64 objects in total in each of the images.

Chapter 4

Empirical Study

The study performed in this work presents a series of carefully generated synthetic datasets that were tested on a CNN based model described in Section 3.1. The experiments test the ability of the network to count the number of objects of a certain type in an image in a closed-set classification approach successfully.

This chapter details the evolution of the datasets as the study progressed along with the performance of the network on said datasets. Each iteration of the dataset and its effect on the output of the network is explained with the help of confusion matrices and activation maps of the final convolutional layer. These performance metrics form the basis of the decisions on how to update the dataset in a way that positively influence the ability of the network to count. Table 4.1 gives a primitive overview of the performance of the different networks on each iteration of the dataset. These numbers were used only as a guideline to evaluate the performance of the network.

4.1 Simpler Incarnations of the Counting Problem

Section 3.3 defines the final version of the dataset that has images of a resolution of 448x448 pixels with triangles as the objects. The study started with much simpler datasets that are described in the following sections.

4.1.1 Dots Dataset

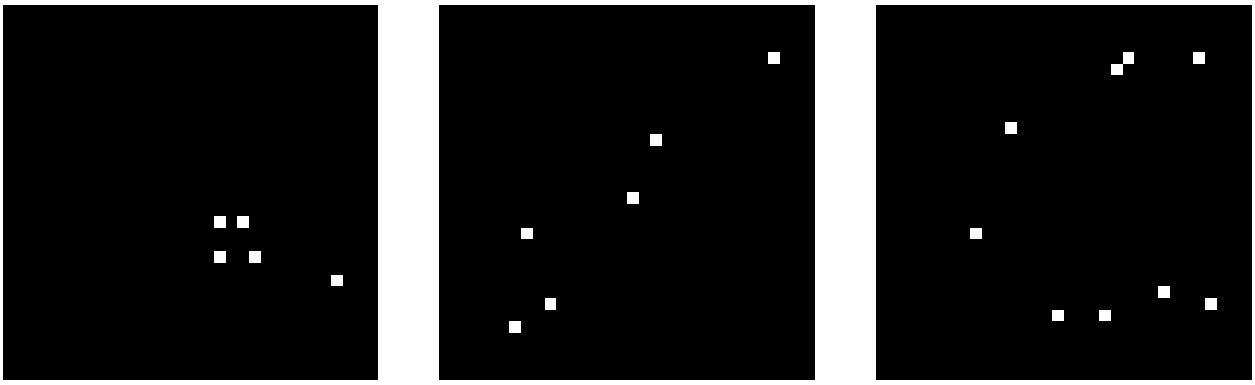
To get a rough baseline on whether or not a neural network can count, a simple dataset in which 32x32 pixels black and white images with zero to 9 dots scattered randomly was created whose samples can be seen in Figure 4.1. The dataset contains a total of 5000 images with 500 samples for each class. The dataset was first tested with a simple fully connected neural network classifier to get a baseline. The network had a flattened input layer of 1024 units and a single hidden layer of 512 units followed by a dropout layer with a dropout rate of 40%. The output layer consists

Table 4.1: Accuracy values obtained by evaluating the network on the test sets of each iteration of the dataset. The *Accuracy* column is the accuracy where the predicted label has to match the target. The *Accuracy - 1* and *Accuracy - 2* columns allow an error of ± 1 and ± 2 respectively.

Dataset	Accuracy	Accuracy - 1	Accuracy - 2
Dots (FC Network)	53.07	92.73	99.33
Dots	49.13	89.20	98.47
Circle-Diamond	24.05	64.56	87.97
Circle-Diamond Shifted	24.05	64.56	87.97
3x3 Grid	99.88	99.98	100
3x3 Fixed	29.69	79.69	100
4x4 Fixed	100	100	100
5x5 Fixed	100	100	100
6x6 Fixed	100	100	100
2-Shapes (0% Occ)	99.99	100	100
2-Shapes (10% Occ)	92.44	99.98	100
4-Shapes (0% Occ)	99.86	100	100
4-Shapes (20% Occ)	86.34	99.92	99.99
3x3 Fixed (Max Pool)	28.91	82.03	100
2-Shapes (0% Occ) (Max Pool)	26.33	53.02	70.19

of 10 units that classified an image into one of the 10 classes based on the number of dots it has. Following that, the next test was with the CNN based network with small modifications made to the base architecture described in Section 3.1. Those included changing the input to be of the shape 32x32x1 and the final dense layer to have 10 units.

The Dots dataset was used as a baseline to gauge how does a neural network count in a *"closed set classification"* approach. Table 4.1 shows the performance of the two networks that were tested on this dataset. The fully connected network does not fare well on this dataset. This was especially concerning as this dataset was found to be linearly separable with respect to the average gray levels, or average intensities, of the images (Figure 4.2). The fully connected network, in theory should have had no problem with this dataset. A similar result was obtained when the CNN based model was tested with this dataset. Unlike the fully connected network, the CNN based model generates metrics that are better to understand and make decisions upon, i.e. activation maps. Figure 4.3 shows what the last convolutional layer looked at when Figure 4.1b from the test set was passed as



(a) Count–5. **(b)** Count–6. **(c)** Count–9.

Figure 4.1: Sample images from the baseline Dots dataset.

an input to the network. This kind of fuzzy and low-resolution activation could be explained due to the relatively small size of the images and even smaller size of the objects that the network is supposed to look at. Due to this, the next version of the dataset has its images generated in a higher resolution than 32x32 and the small dots are replaced with larger solid circles.

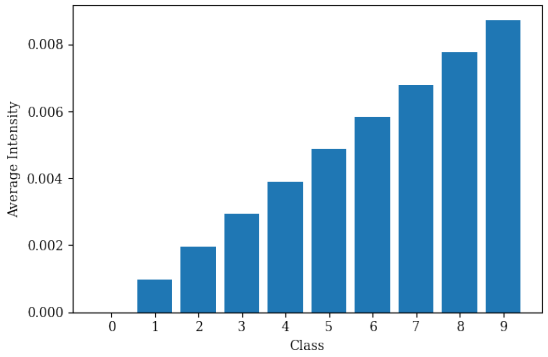


Figure 4.2: Average intensity (gray level) per class for the Dots Dataset.

4.1.2 Circle-Diamonds Dataset

Section 4.1.1 concludes with an updated requirement for the dataset that the network needs higher resolution images than the Dots dataset. To that end, the new images had a resolution of 224x224. To fix the issue of the objects being too small for the network to properly recognize, the dots were replaced by solid circles. To better manage the image generation process as now

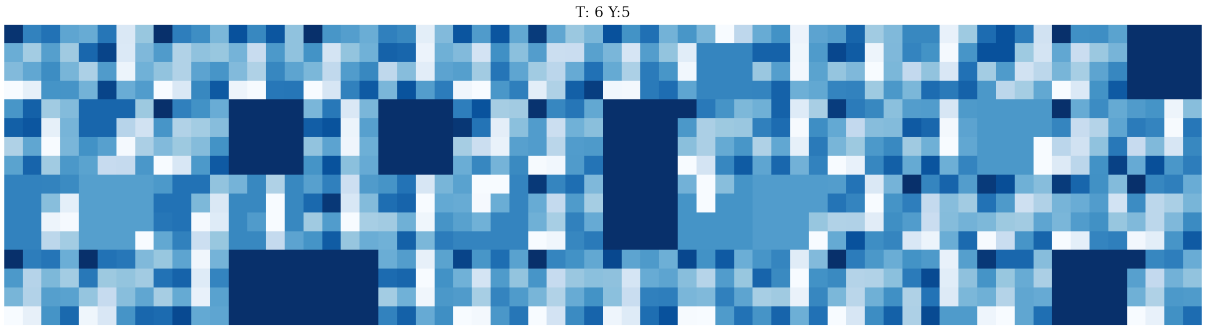
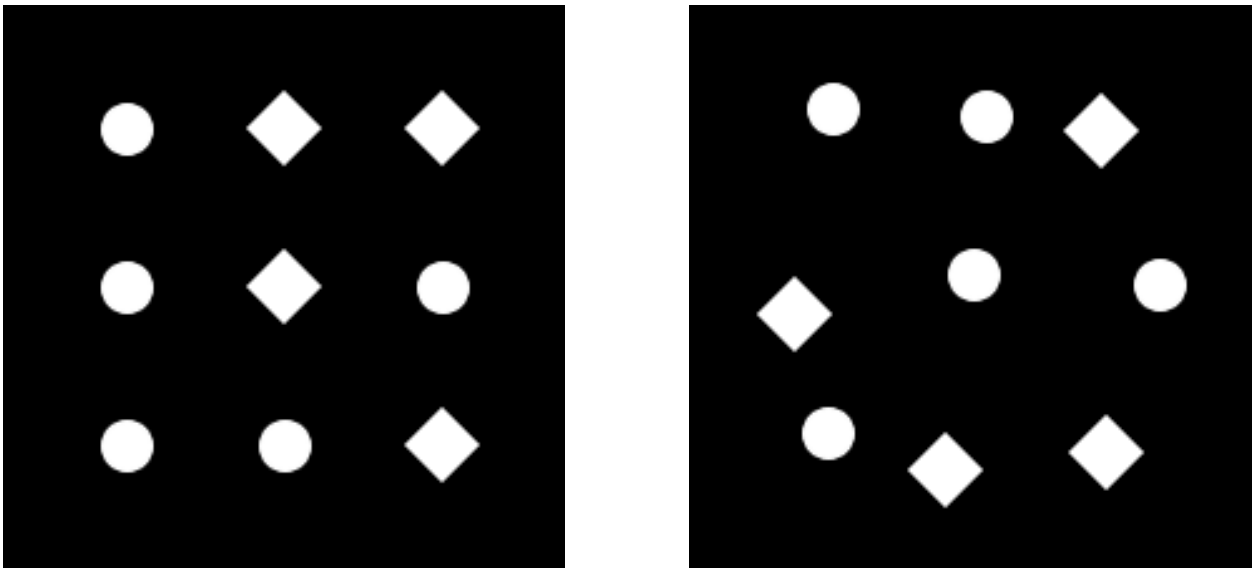


Figure 4.3: Activation map generated by the final convolutional layer for a sample from the Dots Dataset.

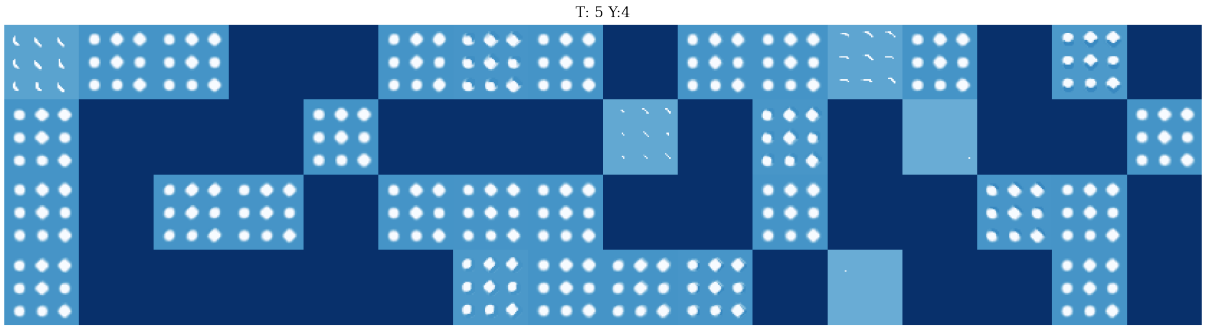
the images had considerable large objects, the circles were arranged in a 3x3 grid instead of being scattered randomly. The images had zero to 9 circles occupying the 9 possible positions. All combinations for possible positions for the circles were used. Solid diamond shapes were added in the empty positions of the 3x3 grid to remove the average intensity gap between the different classes. The fixed grid layout although helped the image generation pipeline, introduced the possibility of the network fixating on certain locations in the image.



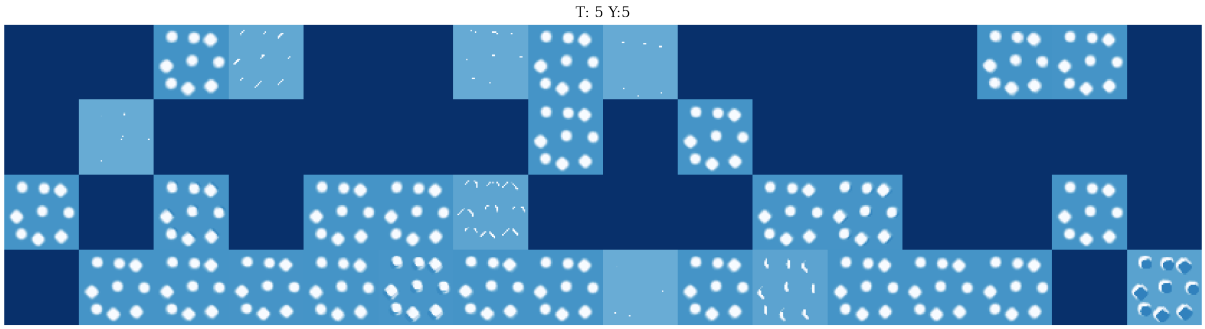
(a) Count–5. **(b)** Count–5.

Figure 4.4: Sample images from the Circle-Diamond dataset. (a) Objects in a fixed 3x3 grid. (b) Objects in a 3x3 grid with centers shifted by a small value.

Two versions of this dataset were created to test whether or not the network only looks at certain positions. In the first version, the objects are fixed in the 3x3 grid as seen in Figure 4.4a and in the second, the objects are shifted by a small random value in a random direction as seen in Figure 4.4b. For reference, these versions will be referred to as the "circle-diamond" and "circle-diamond shifted" datasets. The task here in both these versions of the dataset is to count the number of circles.



(a) Circle-Diamond Dataset.

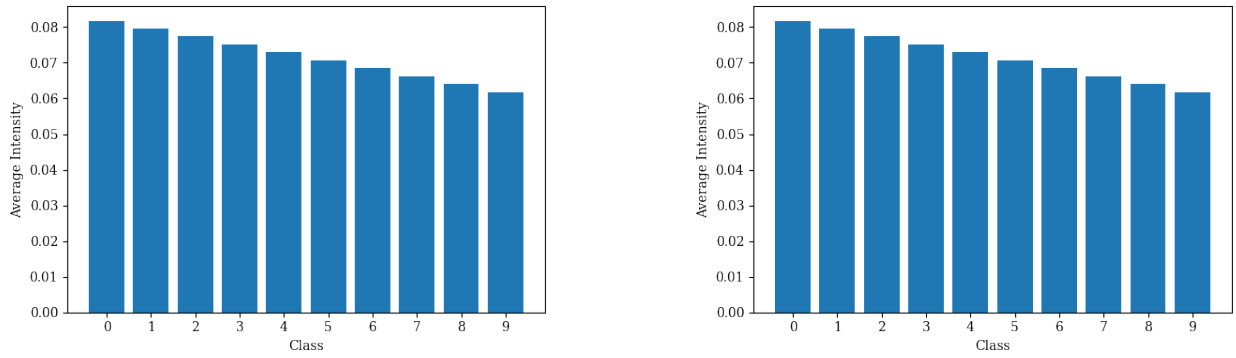


(b) Circle-Diamond Shifted Dataset.

Figure 4.5: Activation map generated by samples from the Circle-Diamond datasets. (a) Map generated by Figure 4.4a. (b) Map generated by Figure 4.4b.

The Circle-Diamonds dataset was created to provide *higher quality* images for the network to be tested upon. Table 4.1 shows sub-par performance by the CNN based model in this dataset despite the changes. Contrary to Figure 4.3, the activation maps for samples from Circle-Diamond are much easier to comprehend; Figure 4.4a and Figure 4.4b generate Figure 4.5a and Figure 4.5b respectively. Both these maps show that the network does identify both objects in the images,

however, that does not translate into a better performing network. This was due to the fact that the network could not differentiate between what it was supposed to count, i.e. circles, and what it was supposed to ignore, i.e. diamonds. A speculation for this behaviour could be made that since the diamonds did not removed the average intensity delta between the classes (Figure 4.6) and the two different objects in the image provided conflicting information to the image.



(a) Circle-Diamond Dataset.

(b) Circle-Diamond Shifted Dataset.

Figure 4.6: Average intensity per class of the Circle-Diamond datasets. (a) Plot for fixed 3x3 grid version. (b) Plot for shifted 3x3 grid version.

The next concern with this dataset was whether or not the network would memorize the positions of the objects as a fixed 3x3 grid was used to generate the dataset. To that end, the Circle-Diamond Shifted dataset was created. Even though the network performed equally bad for both versions of the dataset (Table 4.1), the important takeaway here was that the network performed *equally* bad. The activation map for the shifted version in Figure 4.5b shows similar activations as the regular circle-diamonds dataset in Figure 4.5a which means that the network was invariant to the position of the objects in the image. One specific observation about these maps is that some filters detect the edges of the objects. That means that the network was trying to identify the objects based on the specific shape of the objects. This is an important observation because as the experiment progressed, the dataset is updated to make sure the network separately identifies the objects that it is supposed to count and those it should ignore.

4.2 A Shift to Triangles

The experiment in Section 4.1.2 established that the network does not memorize positions on the 3x3 grid layout and was trying to identify the two different shapes. The thing that was holding it back was the still existing average gray level difference between each class. This proved to be impossible to remove with circles and diamonds as getting both of them to be the same size in terms of area was difficult.

This lead to the next iteration of the dataset where the circles and diamonds were both replaced by triangles that could be created with the same sizes. Now, to differentiate between the objects that the network was supposed to count and ignore, the triangles were oriented in two different directions; upward and downward facing. The task of the network from here on out was to count the number of right-side up triangles in an image. The grid layout was carried over from the Circle-Diamond dataset as the image generation process was easier to control and the network was proved to be invariant to the position of the objects.

4.2.1 3x3 Grid Triangles

The initial version of the Triangles Counting Dataset, referred to as 3x3 Grid Triangles, was generated to overcompensate for the average intensity delta by having zero to 9 triangles in total which were then further divided to be either right-side up or down. These triangles were then arranged in the 3x3 grid layout that was brought over from the Circle-Diamonds dataset and all possible combinations of the positions in the grid were used. Figure 4.7 shows two samples from this dataset. The dataset was tested with the same CNN based network as used for Circle-Diamond dataset.

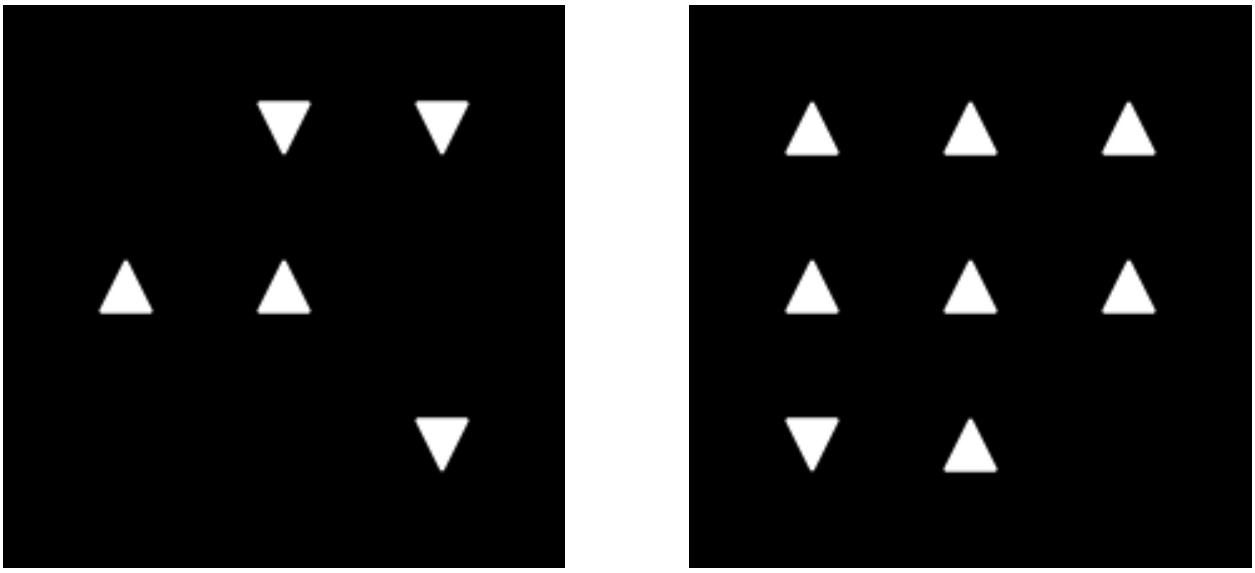
Figure 4.8 shows the activation map generated by the test input image Figure 4.7a with five total triangles and two right-side up triangle among them. This activation map shows the network had learnt to differentiate between the objects to be counted and to be ignored, or in this case, right-side up and down triangles respectively. Majority of the filters show that the network is looking at both, the right-side up and down triangles. Roughly half of them are focusing on the right-side up

Table 4.2: Sample size per class label for initial versions of the Triangles Counting Dataset.

Class	3x3 Grid	3x3 Fixed	4x4 Grid	4x4 Fixed
0	511	1	65535	1
1	2304	9	524288	16
2	4608	36	1966080	120
3	5376	84	4587520	560
4	4032	126	7454720	1820
5	2016	126	8945664	4368
6	672	84	8200192	8008
7	144	36	5857280	11440
8	18	9	3294720	12870
9	1	1	1464320	11440
10	-	-	512512	8008
11	-	-	139776	4368
12	-	-	29120	1820
13	-	-	4480	560
14	-	-	480	120
15	-	-	32	16
16	-	-	1	1

and the other half focusing on the downward facing triangles. This is inferred due to the fact that the objects in focus are clearly identified as the activation takes place for the entire object while the remaining objects are only partially identified, that is, just a corner or part of an edge of the object is causing that portion of the filter to be activated. There are a few filters that exactly identify only the right-side up and downward facing triangles. But on a closer look, they are also identifying parts of the corner(s) edge(s) of the other type of triangle as well. For example, 3rd and 4th from the left on the top row and 7th from the left in the bottom row. There are still quite a few filters that do not get activated at all. And similar to this, even in filters where we see activations, either from the triangles or the background itself, there's zero activations in parts of the image occupied by the triangles.

Now, the triangles were used in place of circles and diamonds because the size could be made the same which would eliminate the average intensity delta between the classes. But due to the



(a) Count–2, Total–5.

(b) Count–7, Total–8.

Figure 4.7: Sample images from the 3x3 Grid Triangles dataset where triangles are arranged in a 3x3 grid with zero to 9 total triangles.

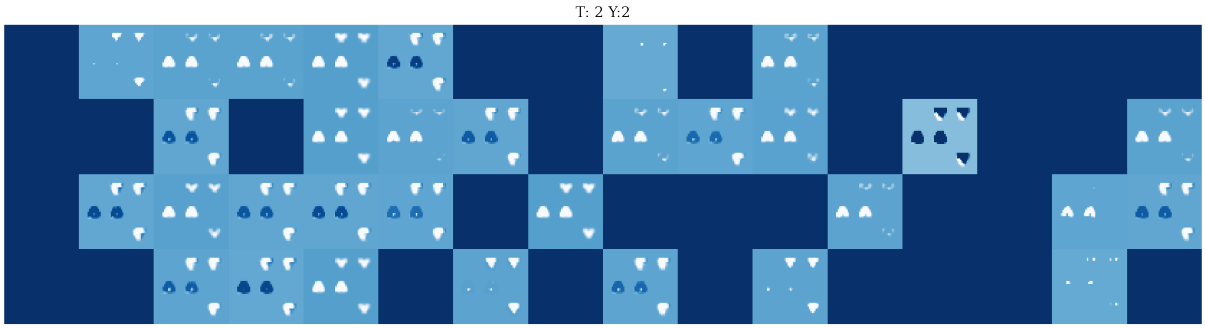


Figure 4.8: Activation map generated for the 3x3 Grid Triangles Dataset from Figure 4.7a.

overcompensation of having all combinations from zero to 9 total triangles inadvertently reintroduced the difference in the average gray levels of the images for each class as shown in Figure 4.8.

One way to fix this difference in the average intensity levels was to update the dataset to be a "two-state" problem wherein a position in the 3x3 grid would either have a right-side up triangle or a downward facing triangle from the current "three-state" problem where the position could be a right-side up or down triangle or an empty space. The gap would disappear as the images would now always contain 9 triangles and both orientations of the triangles are of the same size.

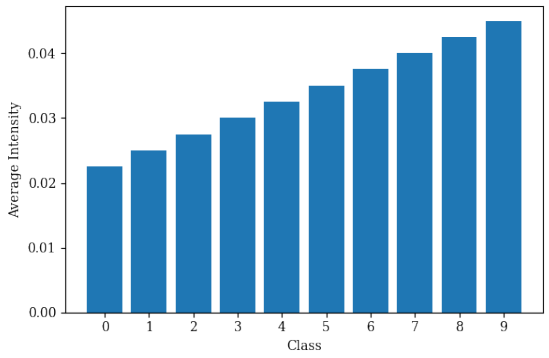


Figure 4.9: Average intensity per class for the 3x3 Grid Triangles Dataset.

A case for increasing the grid size

Column "3x3 Grid" in Table 4.2 shows the distribution of images in each class. A similar dataset but of grid size 4x4 was also considered but was dropped due to the large imbalance in samples per class that would have generated; the column titled "4x4 Grid" gives the class distribution for what could have been 4x4 Grid Triangles Dataset.

4.2.2 Fixed Grid Triangles

The next version of the Triangles dataset again reused the 3x3 grid layout but instead reduced the dataset to *always* have 9 triangles in an image instead of zero to 9 like the previous version. Due to the fact that the images always contained 9 triangles of the same size, the sample size of the dataset was reduced as seen in Table 4.2 under the column "3x3 Fixed". Classes 0, 1, 2, 7, 8, and 9 had fewer samples when compared to classes 3, 4, 5, and 6. To fix this imbalance, those classes were dropped and the experiment proceeded with this reduced version of the dataset. The network structure was the same as before except the final dense layer which now contained four output units.

After dropping the classes that had fewer samples, the dataset did improve its class imbalance, but still had very few overall samples to work with. For that, datasets with 4x4, 5x5 and 6x6 grid layouts were created with the same constraints of always having a fixed number of triangles which

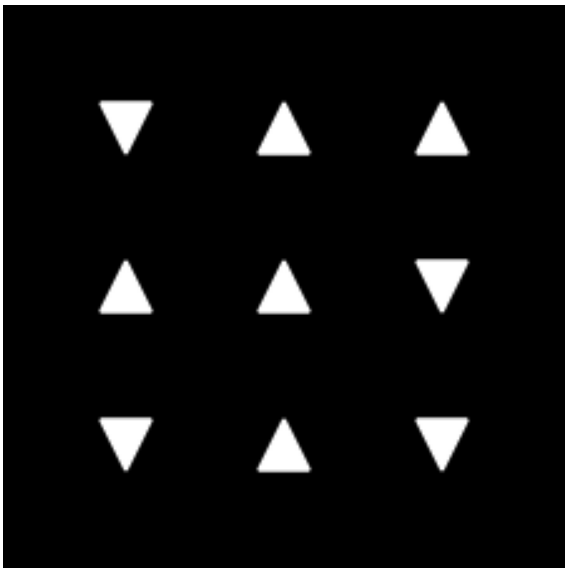
in this case were 16, 25 and 36 respectively. The sample size for 4x4 can be seen in Table 4.2 under the column "4x4 Fixed".

The three new datasets created for this set of experiments of grid sizes 4x4, 5x5, and 6x6 all still had the same issue as the 3x3, class imbalance, which got worse with each step up in the grid size. For this reason, the datasets were also pruned by removing classes with relatively fewer samples followed by under sampling the remaining classes. All in all, the classes that are kept are 3 to 6 for 3x3, 5 to 11 for 4x4, 4 to 21 for 5x5, and finally 3 to 33 for the 6x6 grid. Corresponding changes to the final dense layer were made to the base CNN based architecture used. Samples from these datasets are shown in Figure 4.10. These are also used to generate their respective activation maps in Figure 4.14.

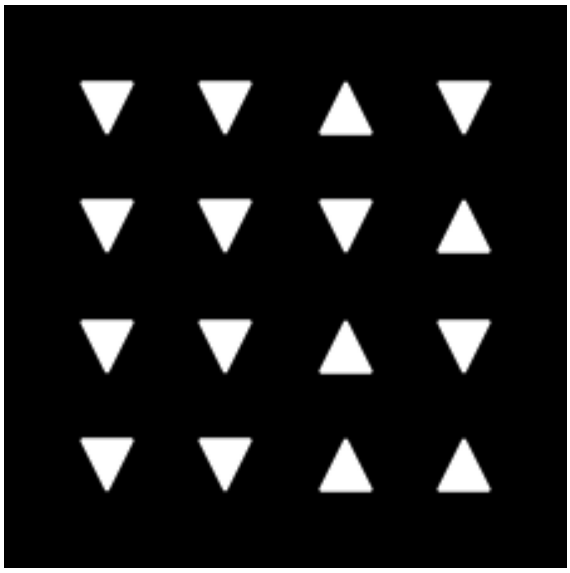
To avoid issues with the average gray levels of the classes, this was tested prior to any experiment and was made sure that the difference was completely eliminated. Figure 4.11a and Figure 4.11b for the Fixed 3x3 and Fixed 4x4 showed that the classes were not linearly separable with respect to average intensity anymore. The same is also true for 5x5 Fixed and 6x6 Fixed Grid datasets.

The evaluation scores for the network on the respective test sets are displayed in Table 4.1. Other than the 3x3 Fixed dataset, all the others have near perfect scores. Figure 4.12 and Figure 4.13 compare the confusion matrices generated by the respective test sets of the datasets that the network was trained with. While the 4x4 Fixed successfully identifies each class, the 3x3 Fixed fails on classes 3 and 6. This may be attributed to the fact that there are less number of samples in those two classes on top of the already fewer number of total samples and distinct classes for this dataset. There isn't much to differentiate classes 3 from 6 and 4 from 5. This issue gets solved when the grid size is incremented as that naturally adds more images for each class.

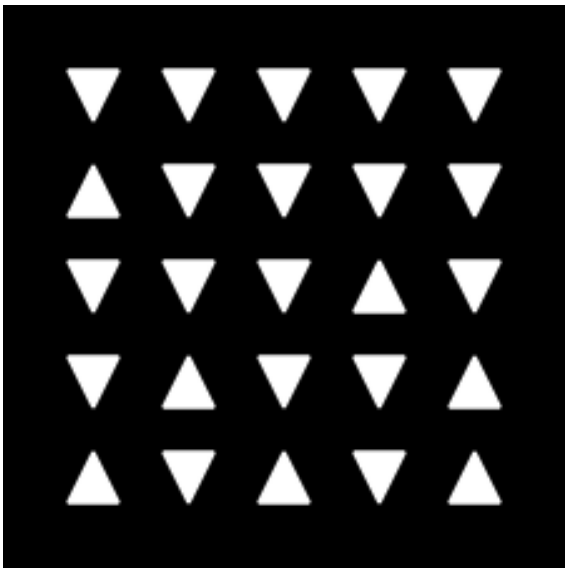
Confusion matrices for Fixed 5x5 and Fixed 6x6 shown in Figure 4.13a and Figure 4.13b respectively show that with a balanced and ample number of images for each class, the network evaluated perfectly with the test set of the dataset. This is also corroborated with the help of the activation maps generated shown in Figure 4.14. As the grid size increases, filters identifying



(a) Count–5, Total–9.



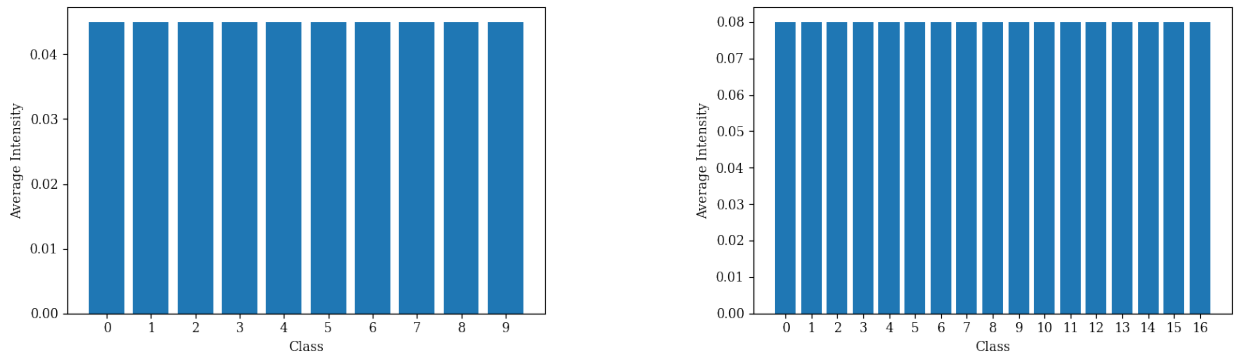
(b) Count–5, Total–16.



(c) Count–7, Total–25.

(d) Count–7, Total–36.

Figure 4.10: Sample images from the Fixed Grid Triangles dataset. (a) 3x3 Fixed Grid. (b) 4x4 Fixed Grid. (c) 5x5 Fixed Grid. (d) 6x6 Fixed Grid.



(a) 3x3 Fixed Triangles Dataset.

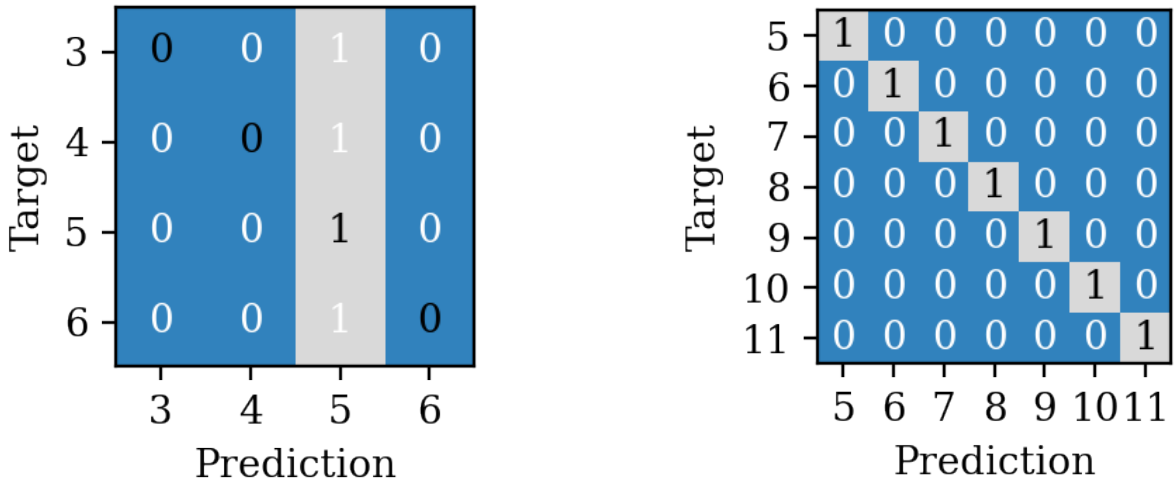
(b) 4x4 Fixed Triangles Dataset.

Figure 4.11: Average intensity per class for the 3x3 and 4x4 Fixed Grid Datasets.

right-side up and down triangles individually also became more robust. Figure 4.14a shows similar issues as Figure 4.5a and Figure 4.5b wherein the network successfully identifies the triangles, but the filters fail to separate the right-side up and downward facing triangles. Some filters, for example, the bottom-rightmost one identified the right-side up triangles.

Figure 4.14b shows a much clear separation between the triangles. Roughly the same number of filters identify each orientation of the triangles individually. This is also true for the 5x5 Fixed and 6x6 Fixed Grid datasets as confirmed by Figure 4.14c and Figure 4.14d respectively. Filters identifying only the edges of the triangles also get fewer in number as the grid size increases showing that the network got better at identifying different triangles. Figure 4.14d however, showed another interesting result, in that, the filters where only downward facing triangles are identified, the activations are *lighter* as compared to the filters identifying right-side up triangles and even the filters from Figure 4.14b and Figure 4.14c that identify downward facing triangles. This may have been due to the cramped nature of the objects in these images.

The ability of the network to successfully count up to 36 right-side up triangles and the fact that these experiments were designed to test the extent to which the network can count led to the decision of increasing the total triangles in the images to 64. The issues that the next dataset should tackle were the cramped nature of the current Fixed 6x6 grid and the increased number of triangles.



(a) 3x3 Fixed Triangles Dataset.

(b) 4x4 Fixed Triangles Dataset.

Figure 4.12: Confusion matrices generated on the test set of Fixed 3x3 and Fixed 4x4 Triangles datasets.

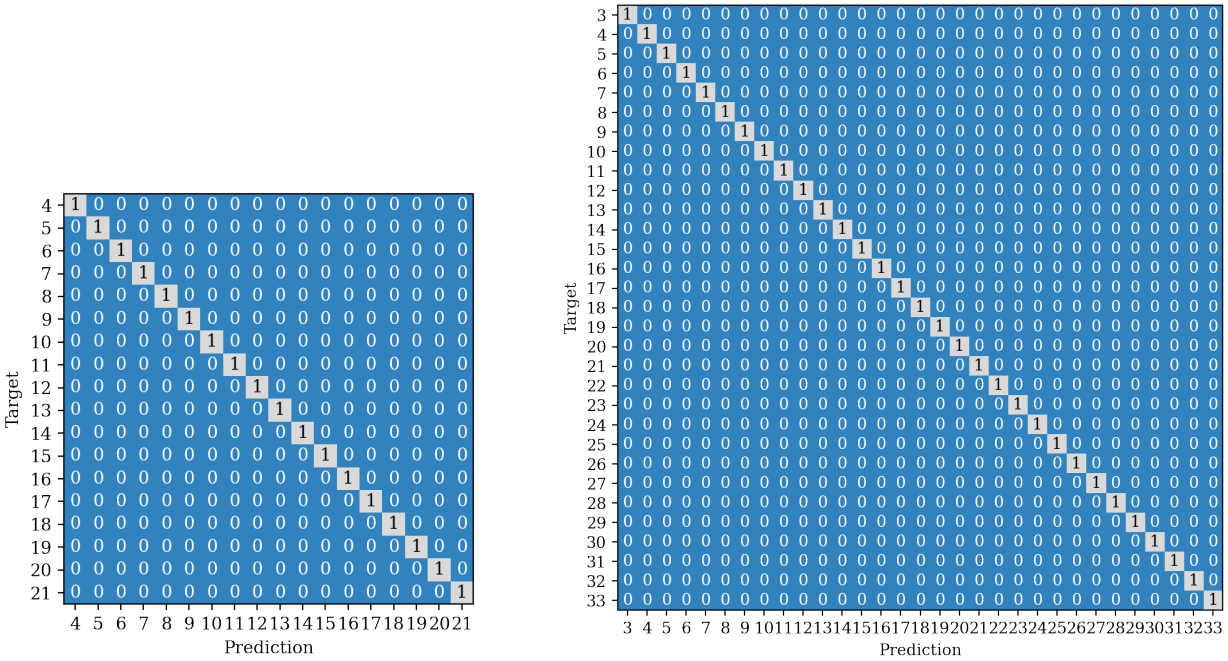
4.3 Final Steps

The next experiment deals with increasing the number of objects in the image to 64. To accommodate these many triangles without the cramping issue as experienced in the 6x6 grid images (Figure 4.10d), the resolution of the images was also increased to 448x448 pixels keeping the triangle size same as before at around 22 pixels wide. In addition to this, the grid layout was abandoned in favour of a random distribution of the triangles.

The results of the network when tested with this dataset which was the penultimate dataset to be tested and the final version of the Triangles Counting Dataset are described in the following sections. The activation maps generated by these images presents a strong argument for the network's ability to count in a closed set capacity even when scaled to have more objects in an image.

4.3.1 2-Shapes — Penultimate Triangles Counting Dataset

The coordinates for each triangle are randomly drawn from a uniform distribution. An image may contain zero to 64 right-side up triangles with the remainder of the triangles oriented facing downward. Getting rid of the grid layout also opened up infinitely more positions that the triangles



(a) 5x5 Fixed Triangles Dataset.

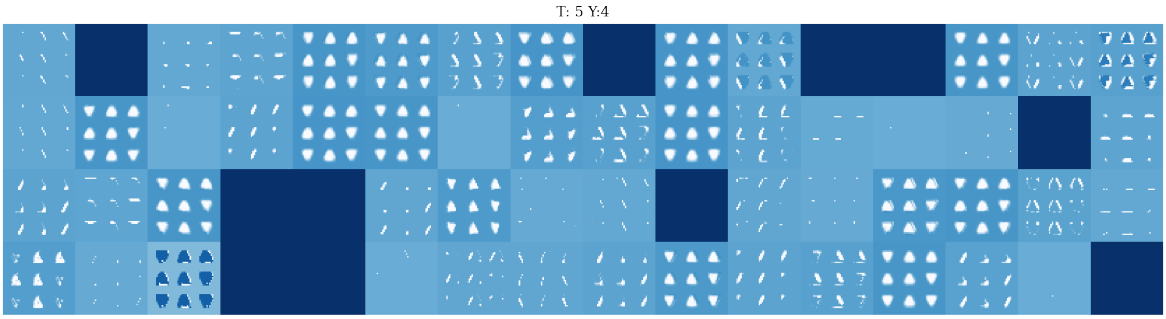
(b) 6x6 Fixed Triangles Dataset.

Figure 4.13: Confusion matrices generated on the test set of Fixed 5x5 and Fixed 6x6 Triangles datasets.

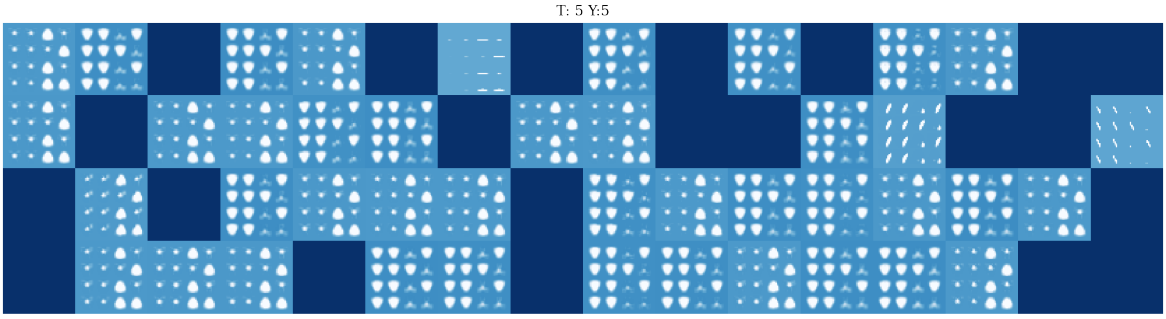
could occupy. With this, 5000 samples per class are generated for a total of 325000 images for the entire dataset.

The triangles are made sure not to overlap each other. But to check the performance of the network properly, another version of this dataset was created that allowed up to 10% of the total number of triangles selected at random to occlude another. Since, this dataset like its predecessors, contains only two orientations of the triangle as compared to the final version, is referred to as "2-Shapes" in this thesis. The goal of this experiment is to test whether the same architecture that was used all this time can handle counting up to 64 even when the objects are scattered randomly instead of a fixed grid.

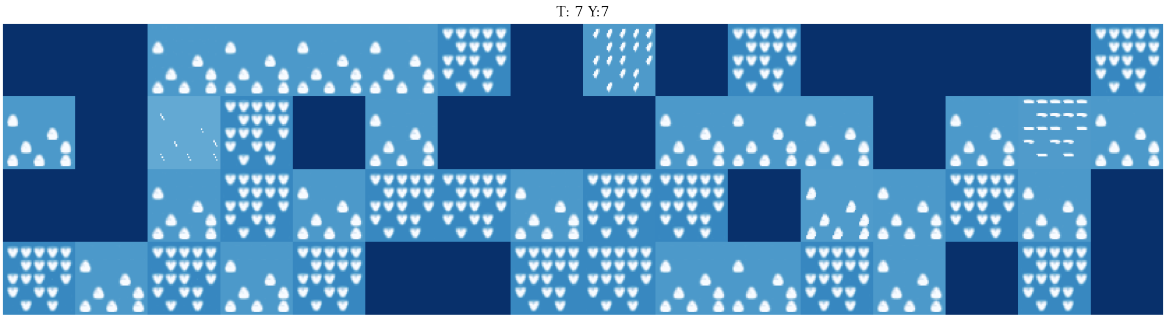
Section 4.2.2 demonstrated the network's ability to filter the right-side up and down triangles and give an accurate count of the number of right-side up triangles. To test the extent to which the network could count, this dataset was created with up to 64 right-side up triangles.



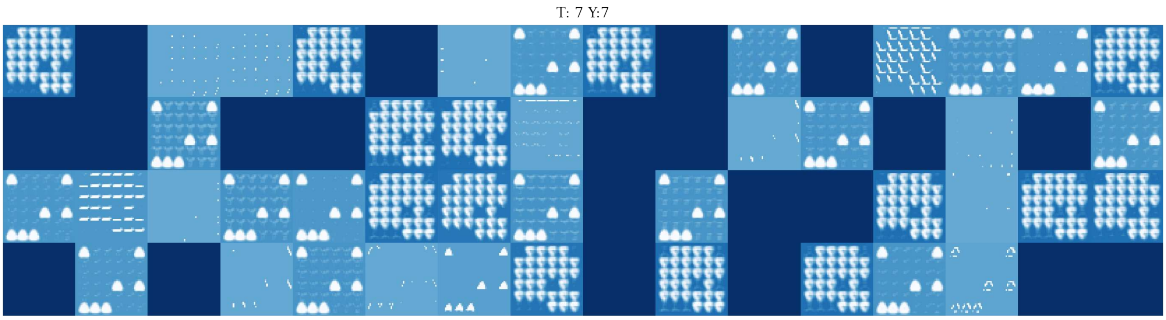
(a) 3x3 Fixed Triangles Dataset.



(b) 4x4 Fixed Triangles Dataset.

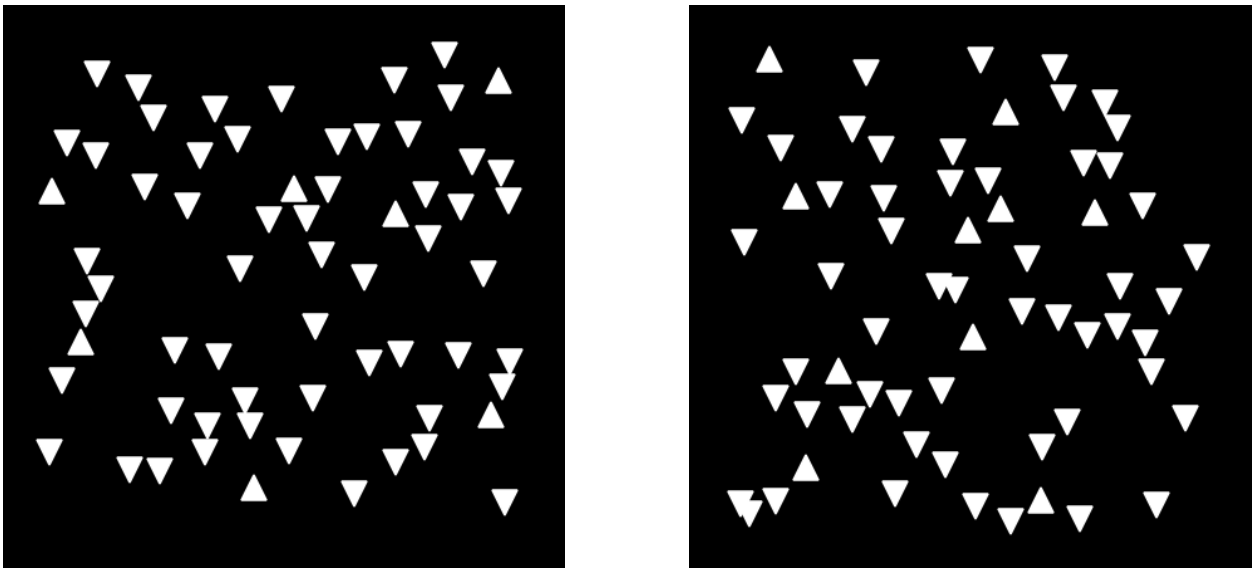


(c) 5x5 Fixed Triangles Dataset.



(d) 6x6 Fixed Triangles Dataset.

Figure 4.14: Activation map plots for all grid sizes of the Fixed Grid Triangles dataset generated by samples from Figure 4.10.



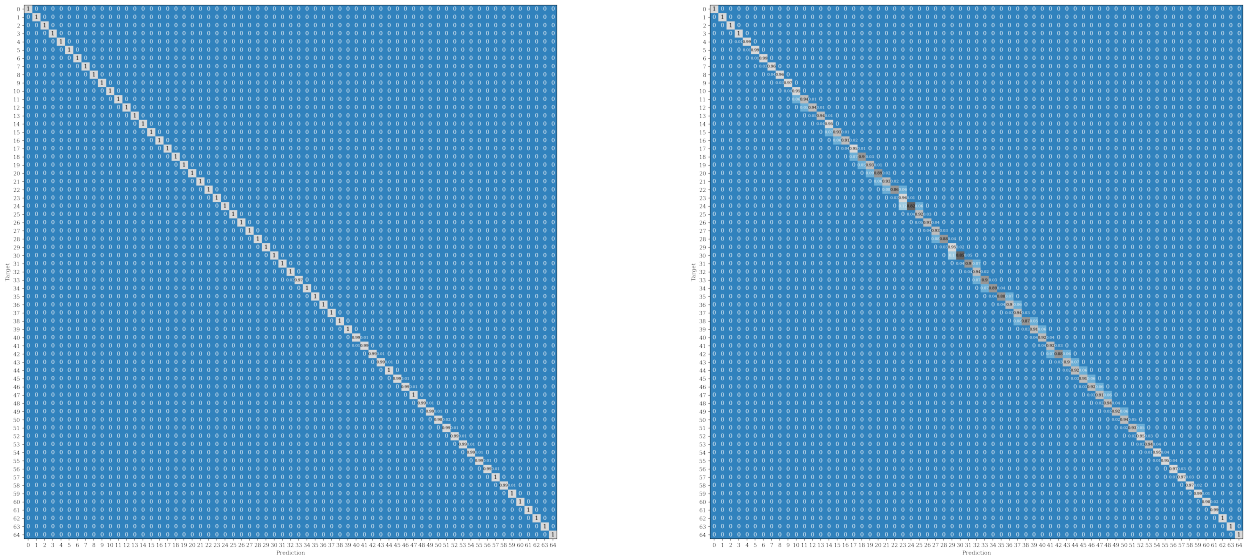
(a) Count–7, Total–64. **(b)** Count–10, Total–64.

Figure 4.15: Sample images from the penultimate 2-Shapes Triangles Counting Dataset. (a) Sample with 0% occlusion between triangles. (b) Sample with up to 10% occlusion between triangles.

Figure 4.17a shows the activations for Figure 4.15a. Here, the filters very clearly individually identify the different triangles. The performance decreases by a very small amount (Table 4.1) when tested with the 10% occluded version as was reflected in the confusion matrix in Figure 4.16b where the network correctly classified all but a few images in the test set perfectly. Still, the activations generated remain largely unchanged, for example Figure 4.15b that generated Figure 4.17b. This experiment confirms with the help of results from all the previous iterations that a neural network can indeed count when presented with an image with up to a fixed number of objects when a closed-set approach is used. The only thing left to confirm was the robustness of this approach when an image contains more than two objects. This lead to the generation of the final version of the Triangles dataset, the 4-Shapes dataset.

4.3.2 Tangent — Testing with a Global Max Pooling layer

At this point in the experiments, it was established that the network architecture described in Section 3.1 was quite capable in counting the triangles in a closed-set classification approach. The feature extractor and the classification layer are joined by a global average pooling that as the name



(a) 0% Occlusion.

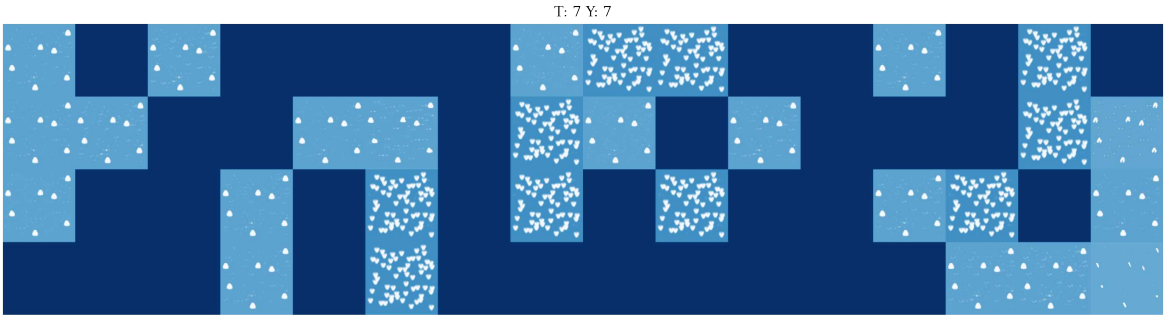
(b) 10% Occlusion.

Figure 4.16: Confusion matrix generated on the test set of both versions of the 2-Shapes Triangles Counting Dataset.

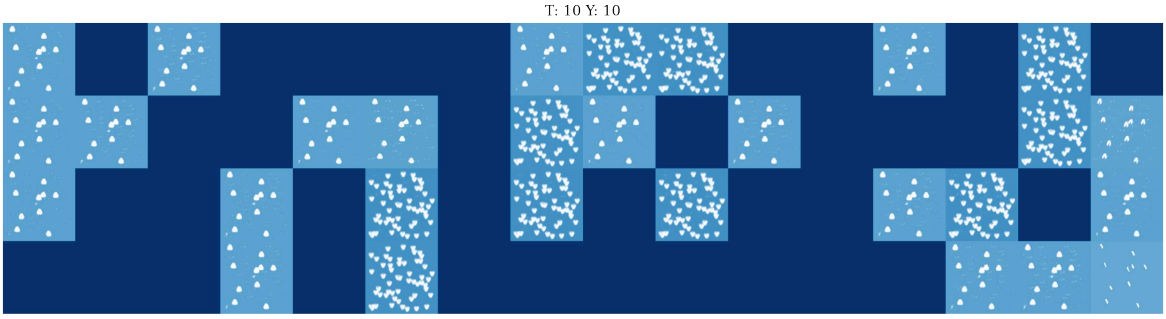
suggests, takes the average of each of the 64 filters of the last convolutional layer to feed into the output dense layer. This tangent explored the effects of swapping this global average pooling layer out for a global max pooling layer. This, unlike calculating the average of each filter, just forwards the maximum value in each of the filter to the next layer.

The modified architecture was then tested on the Fixed 3x3 Grid and the 2-Shapes dataset with rest of the experimental setup being kept the same as when the base architecture was tested with them. Figure 4.18 along with Table 4.1 shows how the network fared with each of these datasets in a purely numerical fashion.

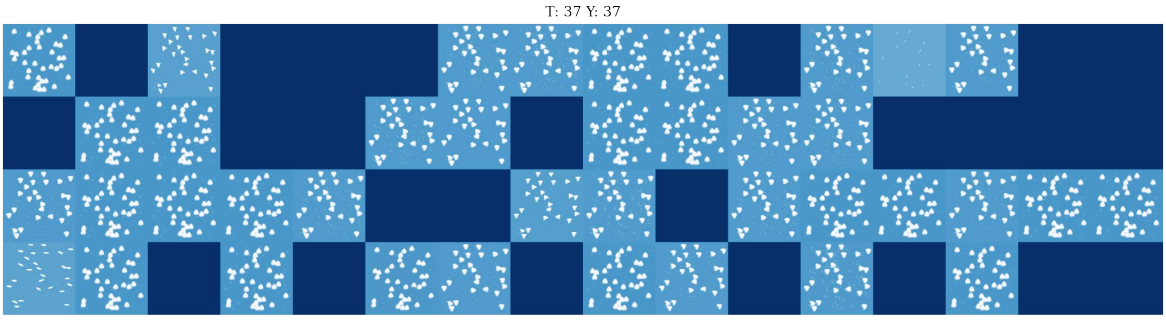
The activation maps in Figure 4.19 show very different kinds of result for samples from the Fixed 3x3 Grid and 2-Shapes datasets. In Figure 4.19c, the filters seem to separate the body of the triangles from their edges where the sample (Figure 4.19a) was taken from the Fixed 3x3 Grid. Nonetheless, it does capture the triangles in the image unlike the activation map for 2-Shapes in Figure 4.19d where, barely any legible activations can be seen. Half of the filters don't produce any activations and only a couple of filters recognize the edges. The rest just show minuscule activations where there are corners of a triangle are present. There seems to be some kind of



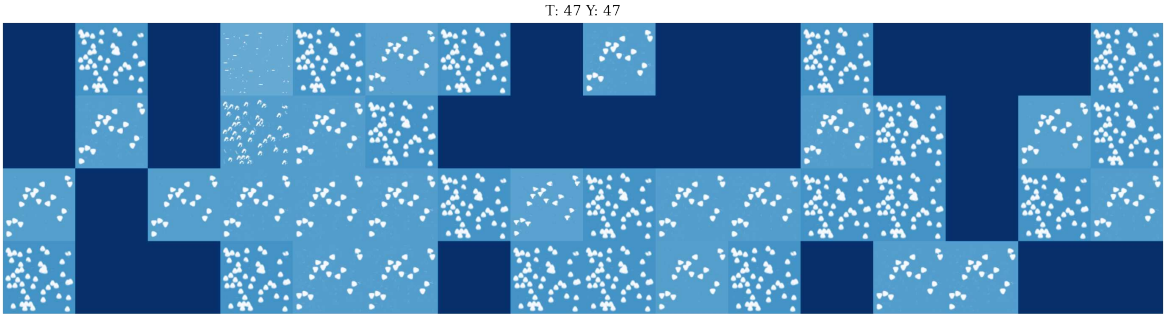
(a) 0% Occlusion.



(b) 10% Occlusion.

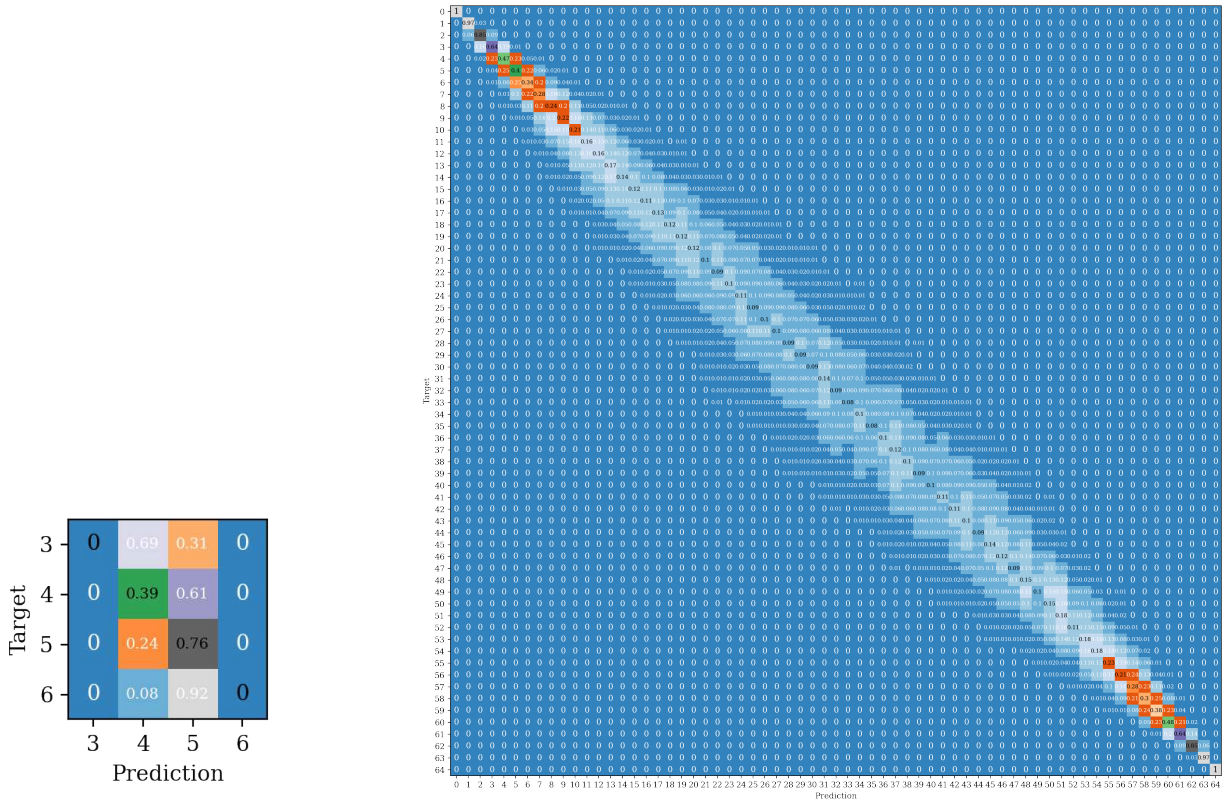


(c) 0% Occlusion.



(d) 20% Occlusion.

Figure 4.17: Activation map plots for the final Triangles Counting Dataset. (a) (b) Generated using samples of 2-Shapes from Figure 4.15. (c) (d) Generated using samples of 4-Shapes from Figure 1.1.



(a) Fixed 3x3 Grid.

(b) 2-Shapes.

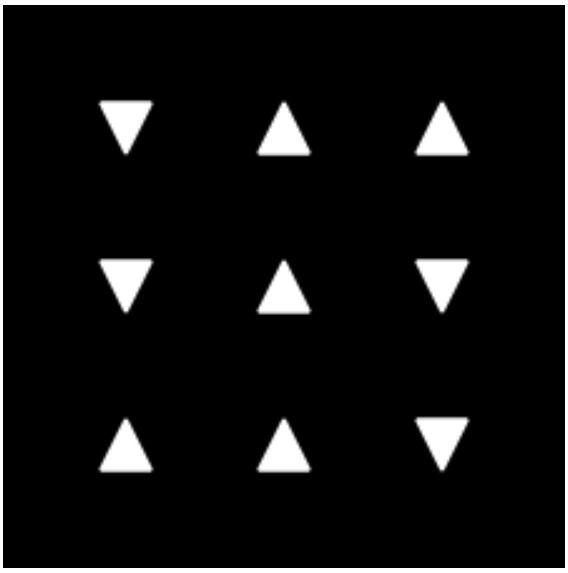
Figure 4.18: Confusion matrix generated by the testing the Fixed 3x3 and 2-Shapes datasets with Global Max Pooling layered network.

separation but the activations are too weak to make any definitive observations based on it. For reference, this activation map was generated using Figure 4.19b.

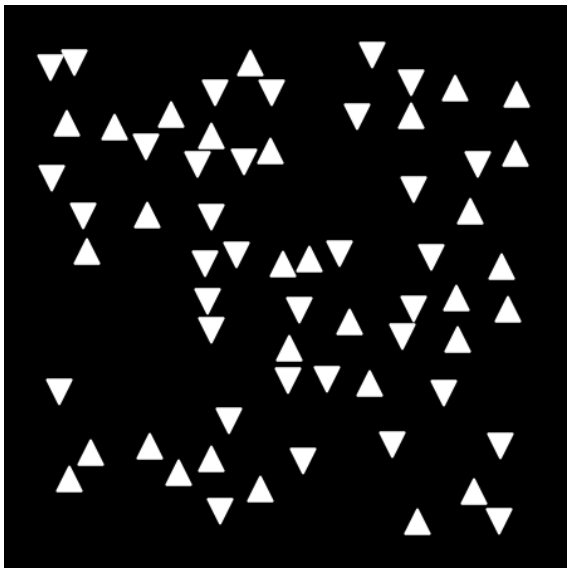
With this mini-experiment, a conclusion can be drawn that for the purpose of counting, a global average pooling layer is the way to go instead of a global max pooling to join the feature extractor to the classification layer.

4.3.3 4-Shapes — Final Triangles Counting Dataset

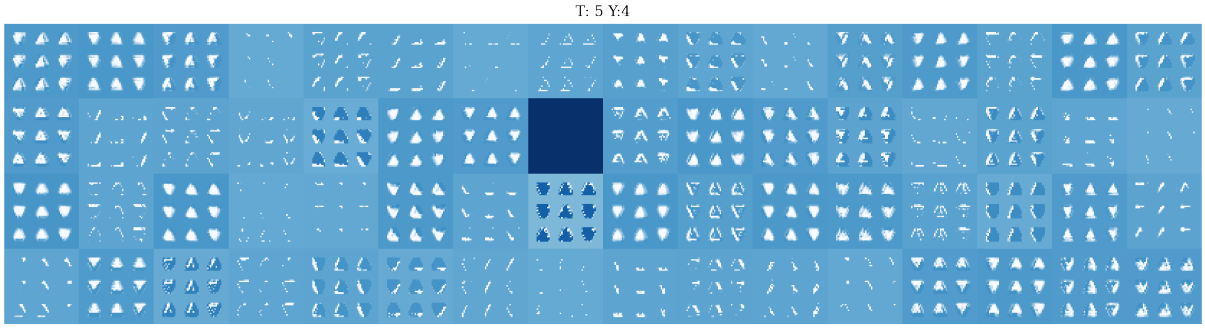
This experiment finishes up the study by tying up the loose ends of the 2-Shapes dataset. The dataset is the same as 2-Shapes in every aspect other than the number of object types. Instead of the two orientations of triangles used in all the previous experiments, this dataset raises that number to 4; namely up, down, left and right facing triangles. The exact nature of this version is described in



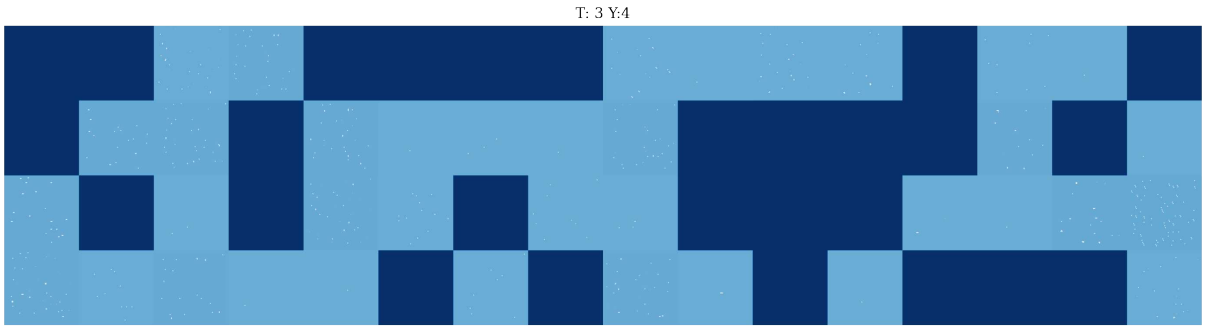
(a) Sample from Fixed 3x3 Grid.



(b) Sample from 2-Shapes.



(c) Activation plot for Fixed 3x3 Grid Dataset.

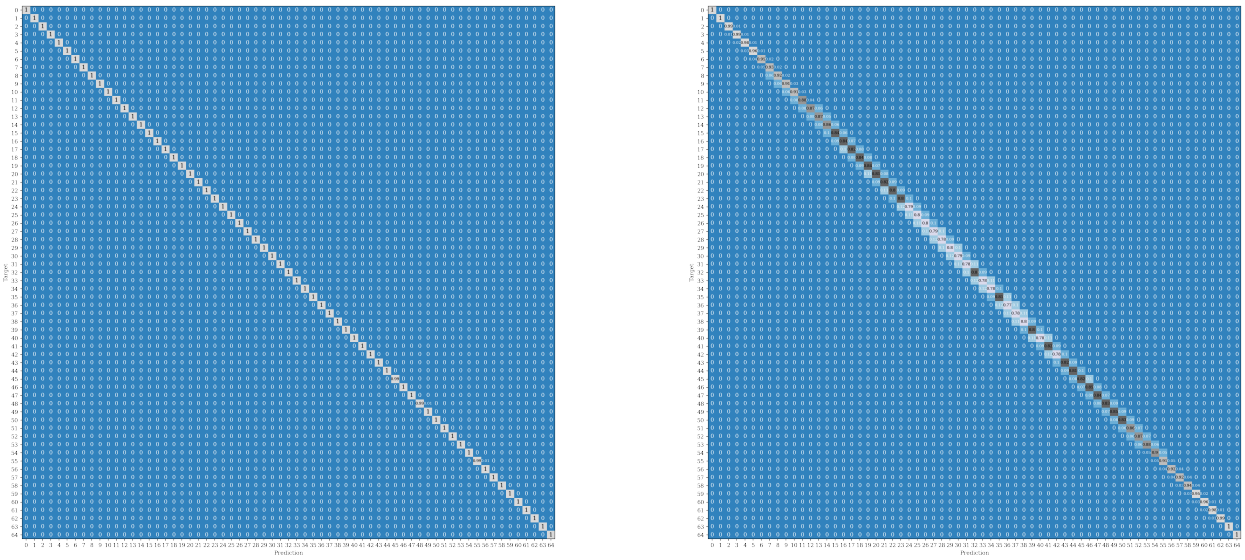


(d) Activation plot for 2-Shapes Dataset.

Figure 4.19: Sample images and the respective activation maps generated by testing the Fixed 3x3 and 2-Shapes datasets with Global Max Pooling network in place of Global Average Pooling layer.

Section 3.3. The performance metrics of this experiment remain unchanged, in that, the confusion matrix and the activation map of the last convolutional layer are recorded. This dataset too, has no linear separability with respect to the average intensity like all the datasets starting from Fixed 3x3 Grid Triangles dataset introduced in Section 4.2.2. Samples from this dataset are showcased in Figure 1.1.

2-Shapes tests the scalability of the network when the number of objects in an image increases. The final version, 4-Shapes, again tests the scalability of the network but in terms of the number of types of objects. The 4-Shapes dataset contains four different triangles oriented up, down, left and right. The task remains the same, that is, count the number of right-side up triangles.



(a) 0% Occlusion.

(b) 20% Occlusion.

Figure 4.20: Confusion matrix generated by both versions of the 4-Shapes Triangles Counting Datasets.

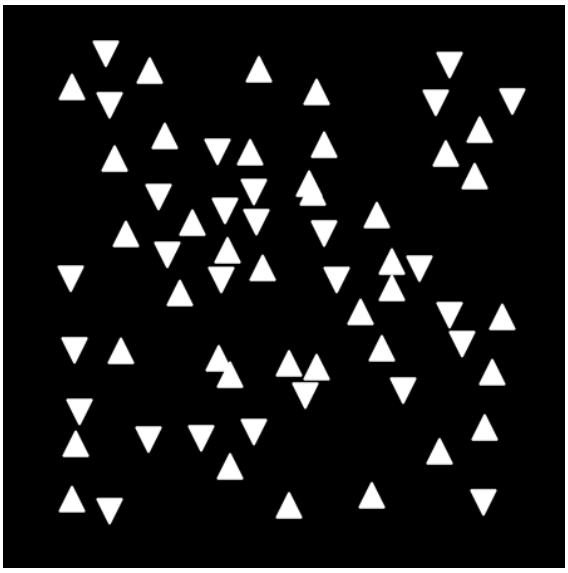
The network maintains its performance when compared to the 2-Shapes dataset which can be seen through Figure 4.20a. The activation map at Figure 4.17c and Figure 4.17d that were generated by inputs Figure 1.1a and Figure 1.1b respectively show that the filters, instead of identifying each of the four orientations separately, identified the right-side up triangles and grouped together the remaining triangles. Asserting the fact that the network indeed looked for the objects it is sup-

posed to count and ignored the remaining. This is even more impressive when taken with the fact that no additional information was provided to the network like the positions (or coordinates) of the right-side up triangles in an image.

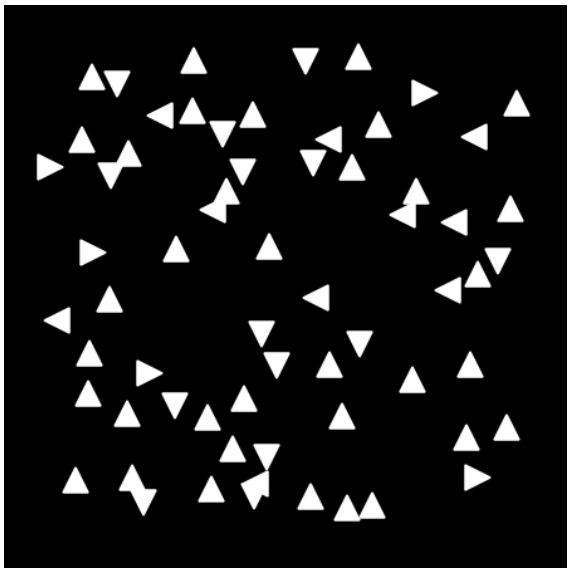
An analysis of the occluded dataset

There was a performance difference between the 0% occlusion version of 4-Shapes and 20% occlusion version of the 4-Shapes dataset. The confusion matrix generated by the 20% occlusion (Figure 4.20b) shows that there is an approximate error of ± 1 triangle for classes 2 to 62. This was unlike the performance on 4-Shapes with 0% occlusion which is almost a perfect classifier. Figure 4.20a shows that the network classifies all but one (class 55) perfectly. The same observation was made after comparing the performance of the 2-Shapes dataset.

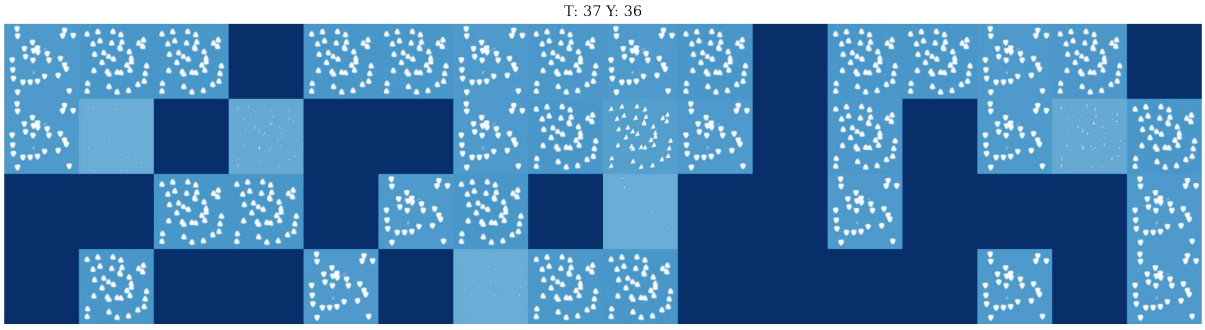
This performance penalty, although minuscule, is due to the occlusion introduced between the triangles. The network does identify the triangles correctly in the last convolutional layer as can be observed in Figure 4.21c and Figure 4.21d. The images that were used to generate these activation maps are Figure 4.21a and Figure 4.21b respectively. This means that the feature extractor does its job perfectly. But since the triangles in the right-side up identifying filters, like the sample images, overlap each other, the global average pooling seemed to get confused when the number of right-side up triangles are more or less than one. From the loss values in Table 4.1, it can be concluded that the network performance weakens as the occlusion and number of shapes (orientations) increase.



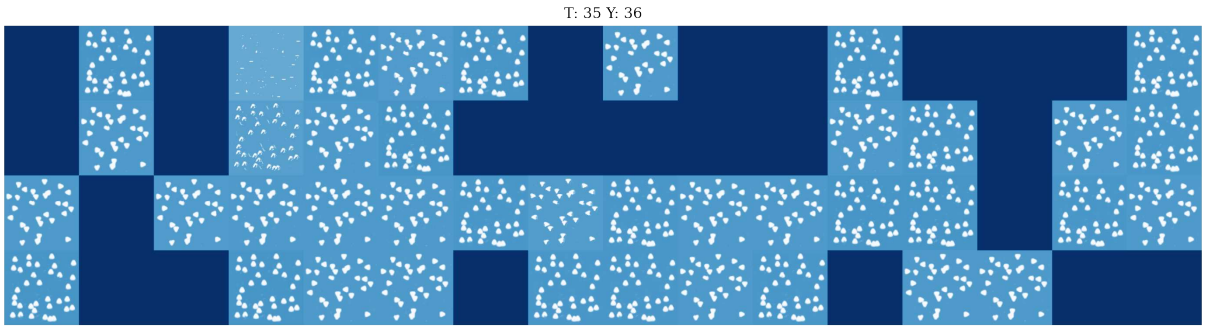
(a) True Count–37, Prediction–36.



(b) True Count–35, Prediction–36.



(c) 2-Shapes Activation.



(d) 4-Shapes Activation.

Figure 4.21: Incorrectly classified images and the respective activation maps from the occluded version of 2-Shapes and 4-Shapes datasets. (a) 2-Shapes with 10% occlusion. (b) 4-Shapes with 20% occlusion. (c) Activation of sample from 2-Shapes. (d) Activation of sample from 4-Shapes.

Chapter 5

Conclusion

CNN based networks are buy and large, the state of the art when it comes to tasks involving images and/or videos. One such task is counting objects in an image. Approaches like density mapping, object segmentation and detection, classification based, etc. are used to achieve specific tasks in object counting like crowd estimation, cell counting, or other practical applications. The work presented in this thesis instead focuses on studying the output of a relatively simple CNN based architecture when tasked with counting. This was achieved through performing controlled experiments on carefully constructed synthetic datasets that allowed to test various aspects of what a network can learn when counting. To strictly *count* instead of *estimate*, the approach taken here was to classify an image in a closed-set manner based on the number of "countable" objects in it.

The synthetic dataset, named Triangles Counting Dataset, consists of carefully generated black and white images. The current version has 448x448 pixel images with approximately 22 pixel wide white triangles scattered throughout a black background. These triangles are oriented in four directions; up, down, left and right. The images have 64 triangles from which zero up to 64 are facing right-side up and the rest being randomly divided into the remaining three orientations. The task is to count the number of upward facing triangles. This is done by classifying the image in one of the 65 classes, where each class denotes the number of right-side up triangles an image can have.

Through the experiments, the ability of the CNN architecture to count was tested. Even a simple network as used in this work can be trained to count, that too without additional information like coordinates of the triangles in the original image. The network does this by effectively separating the image by the objects it is supposed to count and ignore. The activation maps generated for the final convolution layer show filters individually identifying right-side up triangles and the rest grouped together.

The network was also tested to be scalable, both, in terms of number of objects and the number of types of objects. The closed-set classification approach taken in these experiments gives near perfect scores where the objects are clearly visible and suffers only slightly, in the neighborhood of ± 1, when introduced with occlusion between the objects.

5.1 Future Work

This work tested a very simple CNN based network with a few minor variations in a closed-set manner with the focus being on the final layer activations. More experiments could be conducted to assess how each layer contributes towards the final layer separating the right-side up triangles from the rest. The triangle sizes were kept constant for a particular version of the dataset. Experiments can be done to test the how the network fares when objects of varying sizes are present in the same image.

While the datasets were not designed to compare with popular datasets in counting, for example crowd counting, cell counting, etc. in which the count can go up to the thousands, the Triangles dataset, in its current version, has a maximum of 64 objects. Further extent of this network also needs to be tested along with the seeing how the performance changes when the number of layers is increased or decreased. Whether this type of classification based approach suits for such high density counting, remains to be tested.

Much more work needs to be done to deeply and properly understand how a neural network functions when tasked with counting. This thesis aims to at least partially fill the gap in this knowledge of what CNNs learn when counting objects in an image.

Bibliography

[1] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

[2] Bastian Leibe, Edgar Seemann, and Bernt Schiele. Pedestrian detection in crowded scenes. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 878–885. IEEE, 2005.

[3] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. *Advances in neural information processing systems*, 23:1324–1332, 2010.

[4] Antti Lehmussola, Pekka Ruusuvuori, Jyrki Selinummi, Heikki Huttunen, and Olli Yli-Harja. Computational framework for simulating fluorescence microscope images with cell populations. *IEEE transactions on medical imaging*, 26(7):1010–1016, 2007.

[5] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE, 2008.

[6] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 833–841, 2015.

[7] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 589–597, 2016.

[8] YOUMEI ZHANG, FALIANG CHANG, NANJUN LI, HONGBIN LIU, and ZHENDI GAI. Modified alexnet for dense crowd counting. *DEStech Transactions on Computer Science and Engineering*, (cii), 2017.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[10] Jiwei Chen, Wen Su, and Zengfu Wang. Crowd counting with crowd attention convolutional neural network. *Neurocomputing*, 382:210–220, 2020.

[11] Mauro dos Santos de Arruda, Lucas Prado Osco, Plabiany Rodrigo Acosta, Diogo Nunes Gonçalves, José Marcato Junior, Ana Paula Marques Ramos, Edson Takashi Matsubara, Zhipeng Luo, Jonathan Li, Jonathan de Andrade Silva, et al. Counting and locating high-density objects using convolutional neural network. *arXiv preprint arXiv:2102.04366*, 2021.

[12] Meng-Ru Hsieh, Yen-Liang Lin, and Winston H. Hsu. Drone-based object counting by spatially regularized regional proposal network. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[13] Nicolai Häni, Pravakar Roy, and Volkan Isler. Apple counting using convolutional neural networks. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2559–2565. IEEE, 2018.

[14] Haipeng Xiong, Hao Lu, Chengxin Liu, Liu Liang, Zhiguo Cao, and Chunhua Shen. From open set to closed set: Counting objects by spatial divide-and-conquer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8362–8371, 2019.

[15] Haipeng Xiong, Hao Lu, Chengxin Liu, Liang Liu, Chunhua Shen, and Zhiguo Cao. From open set to closed set: Supervised spatial divide-and-conquer for object counting, 2020.

[16] Junkyung Kim, Matthew Ricci, and Thomas Serre. Not-so-clevr: learning same–different relations strains feedforward neural networks. *Interface focus*, 8(4):20180011, 2018.

[17] Sebastian Stabinger, Antonio Rodríguez-Sánchez, and Justus Piater. 25 years of cnns: Can we compare to human abstraction capabilities? In *International Conference on Artificial Neural Networks*, pages 380–387. Springer, 2016.

[18] François Fleuret, Ting Li, Charles Dubout, Emma K Wampler, Steven Yantis, and Donald Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108(43):17621–17625, 2011.

[19] Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Appendix A

Important Links

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

https://python-pillow.org

https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit

https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html