Exploring Edge Detection Methods for Improved Image Classification

Advisor:
Professor Andrea Arnold

Written By:
Juliette Spitaels

May 2023

A Master's Capstone Project
Worcester Polytechnic Institute

Submitted to the Faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Master of Science in Applied Mathematics.

Abstract

This work explores the best practices for edge detection in the context of a deep learning method of quantitative analysis, that is, using neural network accuracy as an objective measure of edge detection performance. To do this we first coded and implemented a variety of popular edge detection methods with an average-based automatic threshold selection algorithm. Then we specifically explored whether blurring an image to reduce noise and false edges is a beneficial step in edge detection in the context of image classification using neural networks. Additionally we tested if applying histogram normalization was a better alternative to blurring in the context of machine learning. Finally, we investigated the different effects of type I and type II errors in edge detection methods in the context of deep learning.

The results indicated that in the context of simpler classification problems blurring did not yield significant improvement to neural network accuracy, but histogram equalization did improve network accuracy. In the context of more difficult classification problems, with more categories and fewer images per category, these results were no longer supported by the data. We also concluded that false-positive edges are preferable to false-negative edges when preprocessing images for classification using deep learning. Finally, we concluded that neural networks could be a valid method for quantitatively assessing edge detection methods, but with the drawback that randomization in the learning process does not guarantee a standard result, and this method therefore requires repetition to manage the variation in results.

Acknowledgements

First I would like to thank Professor Andrea Arnold for her continued support on this project; without her guidance and expertise, this work would not have been possible. I would also like to thank Worcester Polytechnic Institute's Department of Mathematical Sciences for their continued support throughout my educational journey.

Contents

List of Figures

List of Tables

1. Introduction

Image processing is a vast field of study, one which explores techniques for modifying images for a variety of purposes [1]. Many image processing techniques are simply a mathematical operation designed and applied with the goal of highlighting key features in a digital image [2]. Image processing is also typically an important step in preparing images for machine learning applications, such as image classification problems [3].

Edge detection is one popular method of image processing. The goal of edge detection is to find the edges of different elements in a digital image, and highlight these important features, while removing other information from the picture [4]. At the most basic level, edge detection can be done by calculating the difference in a pixel value compared to one or more of its neighboring pixels. But more often, in practice, accurate edges are calculated using methods that build off of the idea of a numeric derivative, with one of the most popular and widely used methods being the Sobel method [5]. An edge detection method is most often applied to an image via one or more filters [6]. A filter can be applied to an image repeatedly, in windows across the entire image, in order to retain only the detected edges in the picture [7]. An example of an edge detection method applied to an image is provided in Figure 1.1.

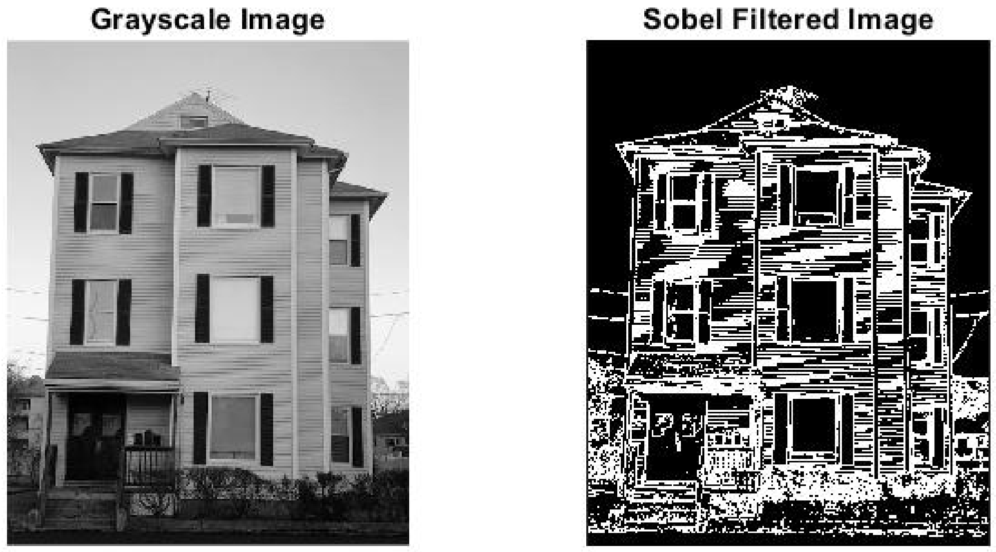

Figure 1.1: Side by side comparison of a grayscale image of a house (left) and the popular Sobel edge detection method applied to the image (right).

Evaluating the performance of an edge detection method is often subjective [8]. Some researchers use the Pratt Figure of Merit in order to evaluate edge detection methods in a quantitative manner, but this method is limited due to requiring a ground truth comparison for ideal edges [4], [9]. Since having an ideal comparison image is rare in practice, this work instead presents an alternative approach of using the accuracy and training time of a simple image classification neural network as a quantitative way of objectively evaluating and comparing edge detection methods.

In order to apply a numeric method, like the Sobel method, it is necessary to determine a threshold such that large enough differences in pixel value are maintained, whereas smaller ones are discarded [10]. But determining such a threshold is a not trivial task; it can be approached with a method as simple as trial and error with subjective, visual evaluation, but this method cannot be easily applied to new images and thus can make evaluating and comparing different methods difficult. The MATLAB image processing toolbox features one automatic threshold method [11], but this paper presents an alternate method that uses the average difference value as the automatic threshold parameter. This thresholding algorithm can be easily implemented as part of different edge detection methods, so they can be fairly compared. This average-based method was found to perform well in many tested scenarios, when considering applications in image classification.

In this work we also explore the best practices of edge detection in context of this alternative method of quantitative analysis, that is, using neural network accuracy as an objective measure of edge detection performance. In particular, we investigate whether blurring was still a beneficial step in edge detection in the context of image classification using neural networks. We further explore the effects of type I and type II errors in edge detection and their resulting effects on a neural network's classification accuracy.

1.1 Contributions

This project contributes a collection of MATLAB functions for a variety of numeric edge detection methods including a central numeric derivative in both the x- and y- directions, Sobel method using 3-by-3 convolution filters, Sobel method using 5-by-5 convolution filters, Sobel method using 5-by-5 filters in the x, y, and diagonal directions, and the Laplacian-of-Gaussian (LOG) method. From these various methods, we report on and compare the computational cost of these algorithms, according to their computation time, and evaluate their effectiveness quantitatively in context of an image identification neural network.

All of the functions also include an automatic thresholding algorithm, based on the average gradient value calculated for an image. This is a valuable contribution since it provides a way to compare different edge detection methods both fairly and easily. We found that in some scenarios this automatic thresholding yielded the best image classification accuracy using a simple neural network, compared to the higher automatic threshold implemented in MATLAB's edge function [11].

The code used to read folders of images and apply functions then write the images to new file locations while maintaining the necessary folder structure was another contribution of this project. All these automatically thresholded edge detection methods were applied to subsets of the Architectural Heritage Elements dataset [12], along with other methods of interest such as Gaussian blurring, histogram equalization, and some methods from the MATLAB edges function [11]. The AHE dataset is a categorical dataset of 64-by-64 images that features a variety of architectural elements such as bell towers and vaults. All of these datasets are structured such that they are compatible with the neural networks used to evaluate the edge detection methods.

The neural networks created for this project are additional contributions. The networks were coded using MATLAB, and more specifically using the deep learning package [13]. The neural networks are compatible with the image datasets created from the AHE dataset [12]. From testing the various datasets via these neural networks we are able to provide quantitative results on the effectiveness of the edge detection methods, which is presented according to the classification accuracy rate and required training time.

1.2 Project Goals

My goals for this project related to both my independent learning and the wider literature about edge detection and its applications. Personally, in this project I aimed to better understand edge detection algorithms and best practices, threshold selection methods, and evaluation metrics. I also worked to build better understanding and skills for deep learning methods, specifically broadening my skill set to include the training and testing image classification neural networks using the deep learning toolbox in MATLAB

[13].

As for the greater context of edge detection, we looked to verify best practices of edge detection, in the specialized context of deep learning. The three main questions we looked to research are:

1. Are neural networks a feasible alternative to figures of merit for quantitatively assessing edge detection methods?

2. Can the traditional step of Gaussian blurring be eliminated when applying edge detection for deep learning applications, while still maintaining good accuracy? Are other preprocessing methods, like histogram equalization, more effective in this context?

3. Are thin edges always preferable over thicker edges in the context of machine learning or can erring on the side of false-positives yield better results?

2. Background

A digital image can be understood as a matrix of pixel values [1]. When an image is defined in this way, it opens up almost endless possibilities for mathematical manipulation of the picture. One of these possibilities is calculating derivatives, or more generally gradients, within the image, which can be interpreted as edge detection [10]. Edge detection is a method of image processing that highlights particular information in a picture. Similar to how grayscale can help emphasize shade in a picture by removing red, blue, and green information (RBG), edge detection helps emphasize distinct elements in an image by removing similarly colored regions, and instead maintaining only the points of high pixel value change [4].

There are many popular methods commonly used for edge detection, but they all have the same objective: identify points of significant change in an image [4]. In turn, this method can reduce the complexity and noise in an image. This method can also reduce the informational density of an image, by replacing many of the entries in the matrix with zero values, making computations with the matrix faster. All of these changes can better prepare an image for use in applications such an image identification with neural networks [3].

2.1 Edge Detection Process

Extensive research into edge detection has highlighted a series of best practice steps for detecting edges in digital images [4]. These include converting images to grayscale, applying a blurring filter, applying the edge detection filter across the entire image, then identifying and applying a threshold for change in the image. This section will describe these steps in detail, as well as explain popular methods of edge detection and further discuss the questions from Section 1.2, as they relate to certain steps in the algorithms.

2.1.1 Grayscale

The first step in a traditional edge detection problem is to convert any color images to grayscale. Some research does explore edge detection in the context of color images [14], but that is outside the scope of this work. A color image has three values assigned to each pixel: one red, one blue, and one green. When imagining a digital image as a matrix of values, a color image is a three-dimensional matrix such that each one of the three colors has its own matrix and each entry corresponds to an intensity of that color for a specified pixel [15]. Converting an image from color (RBG) to grayscale can be done by making a linear combination of the three color values, in order to create one intensity value for each pixel in the grayscale image [10]. The final pixel value will then take on a value between zero and 255, where zero represents a black pixel and 255 represents a white pixel [16]. This scale can also be shifted such that zero represents black, one represents white, and all shades of gray fall within that interval [17]. This is an important step in both the edge detection and machine learning processes, since it significantly reduces the information in the image and allows a picture to be viewed as a one-dimensional matrix [1]. Figure 2.1 shows an application of this algorithm on a sample image.

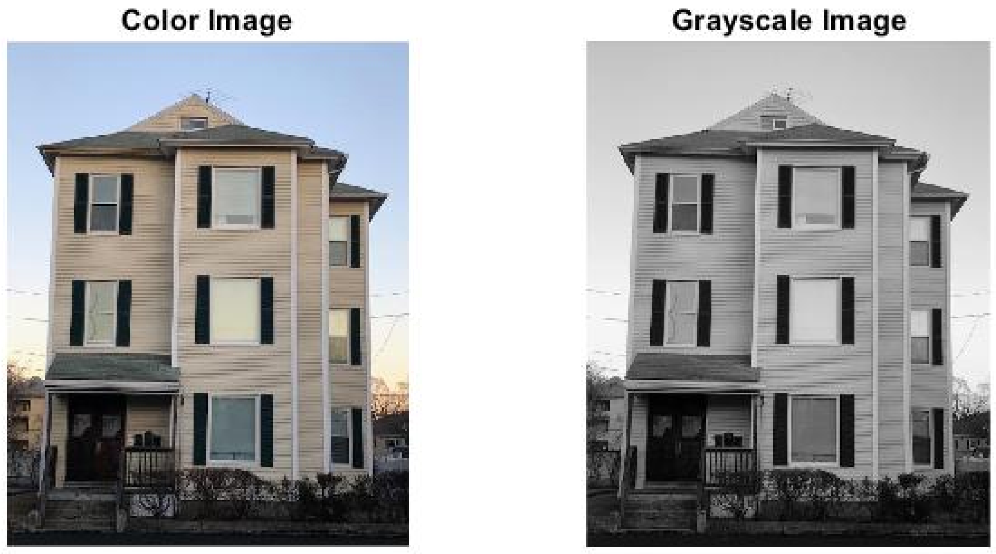

Figure 2.1: Side by side comparison of a color image of a house (left) and a grayscale version of the image (right).

2.1.2 Gaussian Blur

Another step that is often considered best practice in edge detection is to implement Gaussian blurring. This method normalizes pixel values in relation to neighboring pixels to help reduce noise [18]. This is often considered a crucial step in the edge detection process in order to avoid detecting false edges caused by noise in pictures, and it is often applied by default in many edge detection methods [19]. The effects of Gaussian blurring are demonstrated in Figure 2.2.

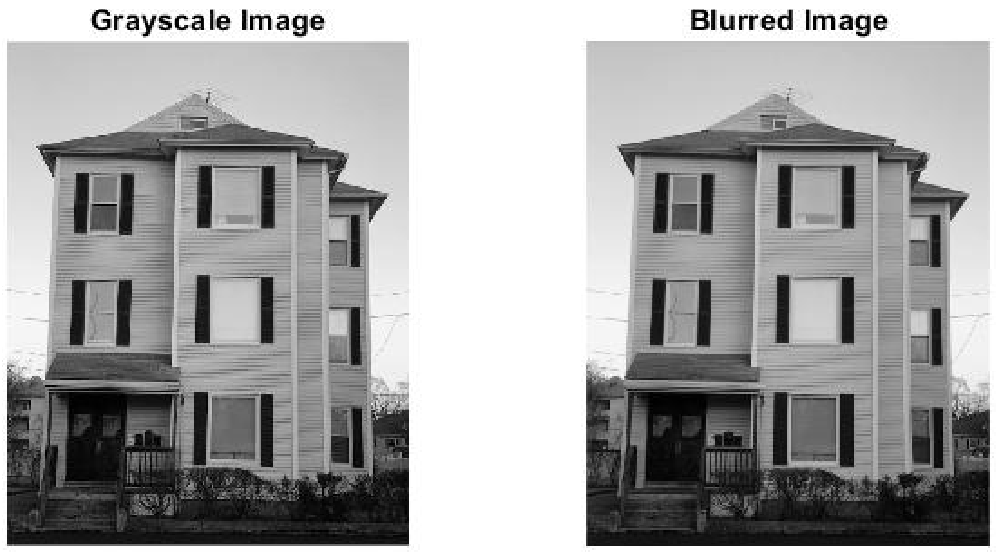

Figure 2.2: Side by side comparison of a grayscale image of a house (left) and a version of the image with the Gaussian blur function applied (right). The differences between the images are subtle but noticeable in elements like the bushes at the front of the house, which are less crisp in the blurred image.

Despite being considered best practice when it comes to traditional evaluation of edge detection, we wanted to investigate the potential of skipping this blurring step when evaluating edge detection methods via image classification results. Significant research has indicated that noise in data can help build more robust machine learning models [20]. In fact, many papers have revealed the benefits of introducing Gaussian noise to a model in order to reduce over fitting in deep learning algorithm and improve testing outcomes [21]. Thus, removing this step from the process and comparing results to traditional reprocessing was one of the areas of exploration in the project.

2.1.3 Histogram Equalization

Histogram equalization is a image processing technique that increases contrast in an image [18]. This method works by analyzing the distribution of pixel values in a given image, which typically follow an approximately normal distribution, and adjusts them to instead follow an equal distribution [22]. This forces more pixel values to the extreme ends of the grayscale spectrum and in turn increases the contrast in the image [22]. Figure 2.3 shows an example of histogram equalization being applied to an image.

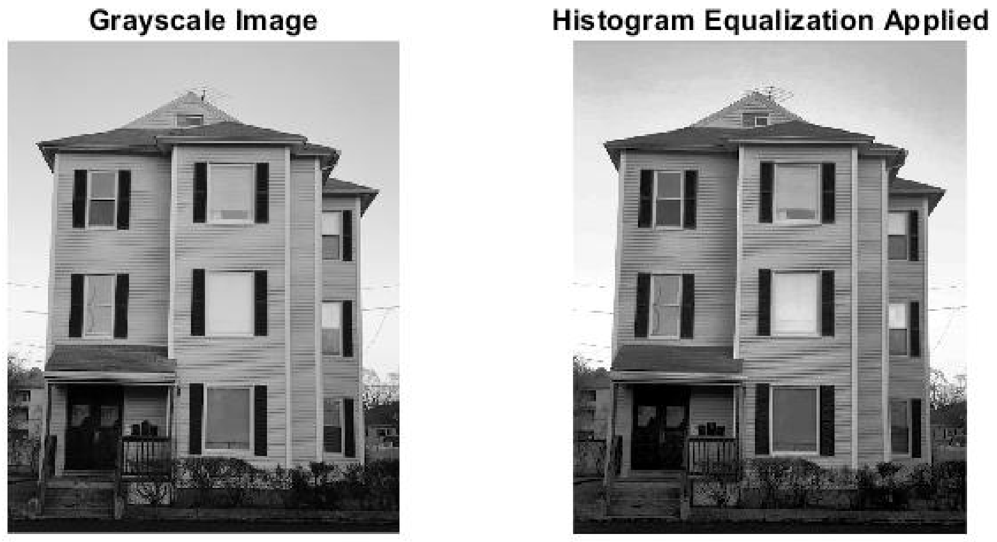

Figure 2.3: Side by side comparison of a grayscale image of a house (left) and a version of the image with histogram equalization applied (right). The difference between the images is subtle but noticeable in the antenna on the top of the house and the siding. Both these elements from the original image become darker after histogram equalization is applied, and contrast more with the lightened sky.

Histogram equalization is not a traditional step in best practices for edge detection, but some research has found positive results from applying this method to images before the step of applying edge detection algorithms [9]. This makes sense, since increasing contrast in an image can also increase the difference between neighboring pixel values, thus making edges easier to detect. Therefore we considered the possibility that histogram equalization could be a better preprocessing step compared to Gaussian blurring when preparing digital images for edge detection and ultimately classification problems using deep learning.

2.1.4 Edge Detection Filter

The next step in the edge detection process is calculating the desired derivatives in an image. These calculations are performed by applying one or more filters to an image via a moving window [7]. The exact filters vary by method, but many of them tie back to an idea of a numeric derivative, so we will use this simple example to illustrate the idea. The edge maps that result from the methods discussed in this section are provided in Figure 2.4.

A numerical derivative is typically calculated by choosing two data points, typically ones near each other, and calculating their difference in the y-direction, then dividing by their difference in the x-direction. In the analytic context, we define the true value of the derivative as the difference between points that are infinitely close to one another. The analytic derivative is represented mathematically such that

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

and thus there is a numeric approximation of the derivative such that

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}. \tag{2.1}$$

The forward difference formula in Equation (2.1), used to calculate numerical derivatives, can be applied to digital images such that each pixel is a data point. In the case of an image, pixel values are evenly spaced, and it is therefore logical to set a simple h value, where we assume the difference between pixel positions is 1. By letting $h = 1$, we get

$$f'(x_0) \approx f(x_0 + 1) - f(x_0)$$

Further, since a large positive or negative change in pixel would both represent an edge, and since pixel values are always non-negative, then it is also logical to apply absolute value to the formula in this application [4]. Thus we can derive an final equation for edge detection such that

$$f'(x_0) \approx |f(x_0 + 1) - f(x_0)|$$

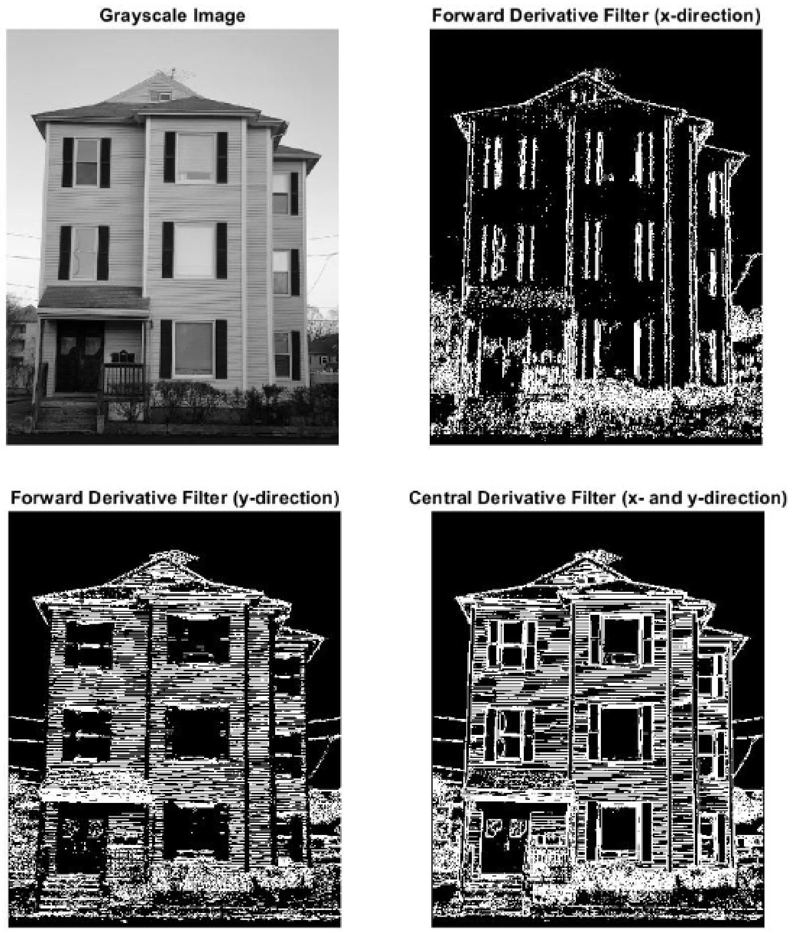
Grayscale Image
Forward Derivative Filter (x-direction)
Forward Derivative Filter (y-direction)
Central Derivative Filter (x- and y-direction)

Figure 2.4: This figure provides examples of three simple edge detection methods based on the forward and central numeric derivative equations. By row from left to right: the grayscale image of a house is provided for reference, the forward derivative method in the x-direction applied to the image, the forward derivative method in the y-direction applied to the image, and a central difference method applied to the image. The x- and y-directions forward derivatives can only detect edges that are perpendicular to the filter, limiting the accuracy of the resulting edge map, whereas the central difference method uses both x- and y-directional filters, and can detect edges in both directions, leading to a more complete representation of the original image.

The effect of taking an absolute value could be achieved equally by applying a 2-norm to the value, that is,

$$f'(x_0) \approx \sqrt{(f(x_0+1) - f(x_0))^2}$$

since squaring and then square rooting the value will always return the positive magnitude.

Applying this formula to a digital image, in the x-direction, can be done by converting the equation for a forward numeric derivative into a simple filter, that is, $\begin{bmatrix}-1 & 1\end{bmatrix}$. It can then be understood that a derivative of zero indicates no change in value, and a derivative value of 255 represents the maximum difference between values, from a white pixel to a black pixel, or vice versa. Then all other potential derivatives that could result from this filter would fall in the range of zero to 255. The same application of the equation could be applied in the y-direction as well, with the filter $\begin{bmatrix}-1 \\ 1\end{bmatrix}$. The process of applying this function to an image can be done via a moving window; that is the operation in applied to the first two pixels in the image, then the filter is shifted by one pixel across the row, and the operation is applied again. When the filter comes to the end of a row, the window is moved to the beginning of the next row and this process repeats until the window has been moved across the whole image, meaning the difference has been calculated for all pairs of neighboring pixels [7]. Visual examples of these different methods applied to the same image are shown as the second and third panels of Figure 2.4.

From this basic example of edge detection in both the x- and y-directions, more complicated filters can be understood. Multiple filters can be combined into a single edge detection method by combining the matrices output by each of the filters. For example, if we modify the previous example to use a central difference method to approximate the derivative in both the x- and y-directions, the associated filters with the method would be as follows:

$$G_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

In this method, each of these filters are again applied in moving windows across the entire image independently, to calculate the derivative in the x- and y-directions, respectively [7]. Then using a 2-norm, the values assigned to each position in the matrix by each of the filters can be combined and given a final gradient value [23]. This final value for each pixel can then be compared to the defined threshold, which will be described next in Section 2.1.5. The demonstration of this method on an image is provided as the last panel in Figure 2.4, alongside the previous methods discussed.

2.1.5 Threshold Selection

Threshold selection is one of the major decisions necessary when implementing an edge detection method. In the case of using a simple derivative method for edge detection, such as those described in Section 2.1.4, the calculated differences from the filter will vary in value on the scale from zero to 255, just like pixel values. But in order to identify an edge, there must be a minimum threshold defined that will be used to classify an edge in an image. If a pixel value exceeds the specified threshold, the value is rounded all the way up to 255 and represented as white; otherwise the value is rounded all the way down to zero and represented as black [24]. It is then clear that a lower threshold allows more details of the image to be preserved, and in some cases a low threshold can also retain noise from the image. A higher threshold puts more of an emphasis on the most significant edges, but once a threshold becomes too high, we can begin to lose some of the vital information in an image as well. An example of the effect that varying a threshold value can have on the final edge map determined for the same image is provided in Figure 2.5.

But as more complicated method are implemented for edge detection, the values assigned to each pixel can take on values with a wider range than just 0-255, due to larger coefficients or multiple filters being used. This starts to complicate the process of choosing not only a useful threshold, but also a threshold that can be applied fairly in order to compare multiple methods.

Oftentimes, thresholds are picked through subjective evaluation or user trial and error [8]. Indications of a good threshold value for a particular image are that the edges are thin, complete, and accurate to the original image [9]. Edges that are not in the original image, but are created through the use of an edge detection method, are considered to be false-positives. Edges that were in the original image but ignored in the edge detection are considered to be false-negatives [4]. Oftentimes false-positives will be called type I errors, while false-negatives are referred to as type II errors [25]. A quantitative method for analyzing edges is the Pratt Figure of Merit, further

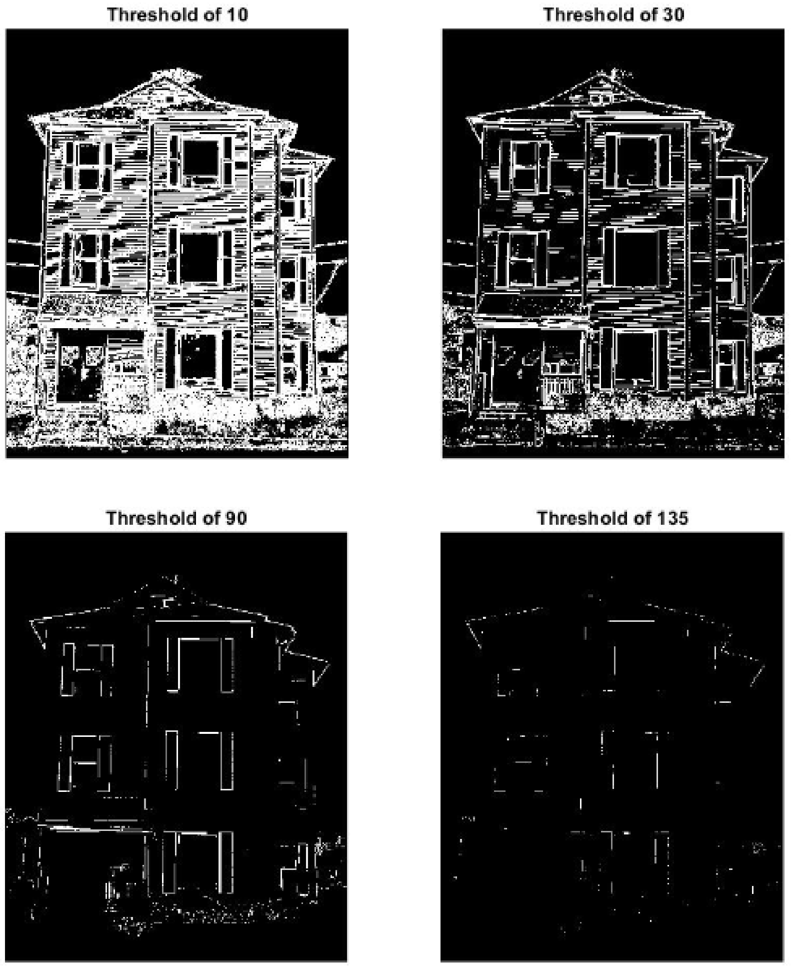

Figure 2.5: This figure illustrates how modifying the threshold for edge detection changes the output of the algorithm. This example uses the a central derivative algorithm explained at the end of Section 2.1.4 which combines both x- and y-directional filters, and tests a low threshold value of then subsequently higher thresholds. By row from left to right: central derivative method with threshold of 10, central derivative with threshold of 30, central derivative method with threshold of 90, and central derivative method with threshold of 135. It is clear from the example that a low threshold maintains more information from the original image compared to the highest threshold, which erases most of the information in the image.

discussed in Section 2.3.1.

Some automatic thresholding methods exist in practice, to varying levels of success. Automatic thresholding approaches using optimization, such as the Otsu method, can be computationally expensive and do not consistently yield good results on varied image types [26]. Other methods, like the default thresholding built into the MATLAB edge function [11], are quick but can also offer inconsistent results, as discussed further in Section 4.2. Since one of the goals of this work is to compare and contrast edge detection methods, we aimed to identify an automatic method for defining the threshold that is both fast and applicable for all methods. Thus we created and implemented a thresholding benchmark defined by the average difference value calculated by a filter. Further detail on this method is provided in Section 3.2.

As we will be comparing this thresholding method to the built-in automatic threshold method used in the MATLAB edge function [11], it is important to detail the function and algorithm in MATLAB, used by default if the user does not specify a threshold. From reviewing the built-in Sobel method function, the first important difference to highlight is the MATLAB algorithm defines a scale factor of 4. Next, similar to our method, they find the average value of change in the image after the specified filters have been applied, but they differ by multiplying by the scale factor, and then taking the square root of that value, which ultimately becomes their automatic final threshold value. This method results in a threshold that is typically higher than the threshold obtained using the method we created for this project. In the case the user would like to specify a threshold for the problem, MATLAB allows an optional parameter for this, acknowledging in their documentation that edge detection is typically a heuristic process [11]. In addition, the MATLAB function differs from our methods since it automatically applies both a blurring filter to the image before processing and an edge thinning method after the threshold has been applied. All these steps together will almost certainly result in fewer edges detected in a given image, as compared to our method.

Moreover, some edge detection methods implement multiple thresholds, which identify both strong and weak edges, then implement additional algorithms to identify which edges to retain and which to discard. The most popular of these methods is the Canny method [19], which will be discussed further in Section 2.2.2.

Despite the difficulty that choosing an appropriate threshold can pose at times, it is crucial in separating the information that an edge detection method will retain in the final image from information that will ultimately be discarded.

2.2 Popular Edge Detection Methods

From the basic building blocks discussed, there are many popular edge detection methods that are used in practice. Two of these that will be discussed in greater detail in this section are the Sobel method [5] and the Canny method [19]. The Sobel method is a method that directly follows the basic method described previously, and this method has been extensively explored and modified by many researchers. The Canny method is a more complicated method that leverages multiple thresholds in order to identify more complete edges. This section will discuss both of these algorithms, and also highlight some other interesting methods.

2.2.1 Sobel Method

The Sobel method is one of the most popular methods of edge detection [5]. The most basic version of this method uses 3-by-3 filters that more heavily weigh the adjacent pixels in an image, and then give less significant weight to the diagonal pixels [26]. This method builds upon the logic of the numeric derivative described in Section 2.1.4, but expands the size and accounts for the additional, often important, information diagonal to a given datapoint. The precise filters used in the Sobel method are provided below, and the difference in weights can be seen in the larger coefficients in the adjacent positions and the smaller coefficients in the diagonal positions [26]:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Then similar to the example described in Section 2.1.4, a 2-norm can be used to combine the results from the two filters into one new matrix representing these derivatives in the image.

Since this method is so widely used, many modified versions of the Sobel method also exist including methods that use alternate coefficient weights [27], larger filters, with 5-by-5 being very popular [28], and even using more than two filters, such as methods that include filters that identify diagonal edges [26]. We further explore the Sobel method and some of the popular modified versions in our experiments, which will be explained in more detail in Section 3.1.

2.2.2 Canny Method

Other methods of edge detection, most popularly the Canny method, begin by filtering an image in the same way as the Sobel method, but differ by adding additional steps, beginning with identify strong and weak edges according to two different threshold values [19]. Through a process called hysteresis, all strong edges are maintained in the final edge map, but some weak edges are also retained, under the condition that a weak edge is connected to a strong edge [19]. These additional steps help ensure more complete edges are identified and kept intact in the final edge map [19]. This method also tends to result in fewer type I and type II errors compared to more traditional methods, such as the Sobel method [29].

The main drawbacks of the Canny method, and other multi-threshold methods that include the hysteresis step, are that the extra steps mean they are more complicated to implement and often more computationally expensive [30]. Therefore, achieving comparable performance to the Canny method from a simpler, single threshold method was one point of interest in this work. In order to quantitatively measure performance of different methods, we use the accuracy of a simple image classification neural network as the metric of comparison.

2.2.3 Other Methods

With the basic method of an edge detection algorithm understood, there are almost endless possibilities for creating new filters. Some filters are created based on mathematical equations, like the Taylor Series [31] or the Laplacian-of-Gauss (LOG) [32]. And some methods are developed for highly specific applications, such as improving clarity in medical scans [31], while others are developed to tackle broader issues in edge detection, like mitigating the effects of noise [32].

The LOG edge detection method uses only a single, 5-by-5 filter, and was designed to incorporate a smoothing effect that minimizes noise in an image while detecting edges, all in one step [32]. The LOG filter is defined as:

$$G = \begin{bmatrix} -2 & -4 & -4 & -4 & -2 \\ -4 & 0 & 8 & 0 & -4 \\ -4 & 8 & 24 & 8 & -4 \\ -4 & 0 & 8 & 0 & -4 \\ -2 & -4 & -4 & -4 & -2 \end{bmatrix}$$

The results of the LOG method, along with many of the other methods described thus far in the paper, will be compared and analyzed in Section 4.2.

2.3 Quantitative Validation

Evaluating different methods of edge detection can be difficult and is often done subjectively [8]. Deciding that one method or threshold is superior to another often includes checking for key characteristics such as thin, but complete edges, which are accurate in placement to the original image [4], [19]. While this may seem simple at first, it is often difficult to make objective, quantitative statements about effectiveness of edge detection methods, especially without a "ground truth" map of edges to use for comparison.

2.3.1 Pratt Figure of Merit

The Pratt Figure of Merit (PFOM) is one of the most widely used methods of edge validation [4]. This figure of merit is an equation that measures edge detection accuracy according to the number of true-positive, true-negative, false-positive, and false-negative edges. True-positive and true-negative edges are when the method correctly identifies an edge, or lack of an edge, in some part of an image. Edges that are not in the original image, but are created through the application of an edge detection method, are considered to be false-positives [33], often called type I errors [25]. Edges that were in the original image but not maintained in the edge detection are considered to be false-negatives [33], or type II errors [25]. The PFOM uses the proportion of true and false edges as one aspect of the calculation to quantitatively assess an edge detection method [4].

The PFOM also accounts for the correct placement of edges in its calculation [4]. Depending on the method and corresponding filter(s) used, edge placement can be shifted or otherwise distorted. If an edge is inaccurate to its location in the original image, this can be a sign of a poor method. Therefore the distance between an ideal edge and the edge created via an edge detection method is also taken into account by the PFOM [4].

Ultimately, the Pratt Figure of Merit method is computed according to the equation

$$PFOM = \frac{1}{max(I_A, I_I)} \sum_{i=1}^{I_A} \frac{1}{1 + \alpha d_i^2}$$

where I_A represents the detected edges, I_I represents the ideal edges, d represents the distance between the actual and ideal edges, and α is a constant used to penalize displaced edges [4]. The resulting value from this function will be in the range [0,1], such that 1 represents a perfect fit to the ground truth. A MATLAB function which compares an image to another specified ground truth, created by Vivek Bhadouria and shared on the MATLAB Central File Exchange [34], can be used to easily calculate the PFOM for an edge map given a ground truth comparison. Note, the function returns values as percents, so in order to have values in the range of [0,1], they must be scaled down by a factor of 100.

The major drawback of the PFOM as a quantitative way of assessing and comparing edge detection methods lies in its need for a ground truth comparison [9]. In practice, very few datasets have ideal edges specified, this process in itself can be biased, and ideals may vary based on the intended use of the edge detection. Sometimes the results of the Canny edge detection method are used as the ground truth comparison, but this assumption is not always fair in the context of all problems, especially in the case of trying to outperform the Canny method with an alternative method. Thus in practice, the PFOM can not always be reasonably used [9]. Figure 2.6 provides an example to illustrate this comparison of our 3-by-3 Sobel method and the MATLAB automatic Sobel method to the MATLAB automatic Canny method.

2.3.2 Neural Networks

As an alternative to a comparative method, such as the Pratt Figure of Merit, which requires a ground truth comparison to derive a quantitative measure of edge detection success, we instead consider using the accuracy of an image classification neural network as an alternative quantitative measure

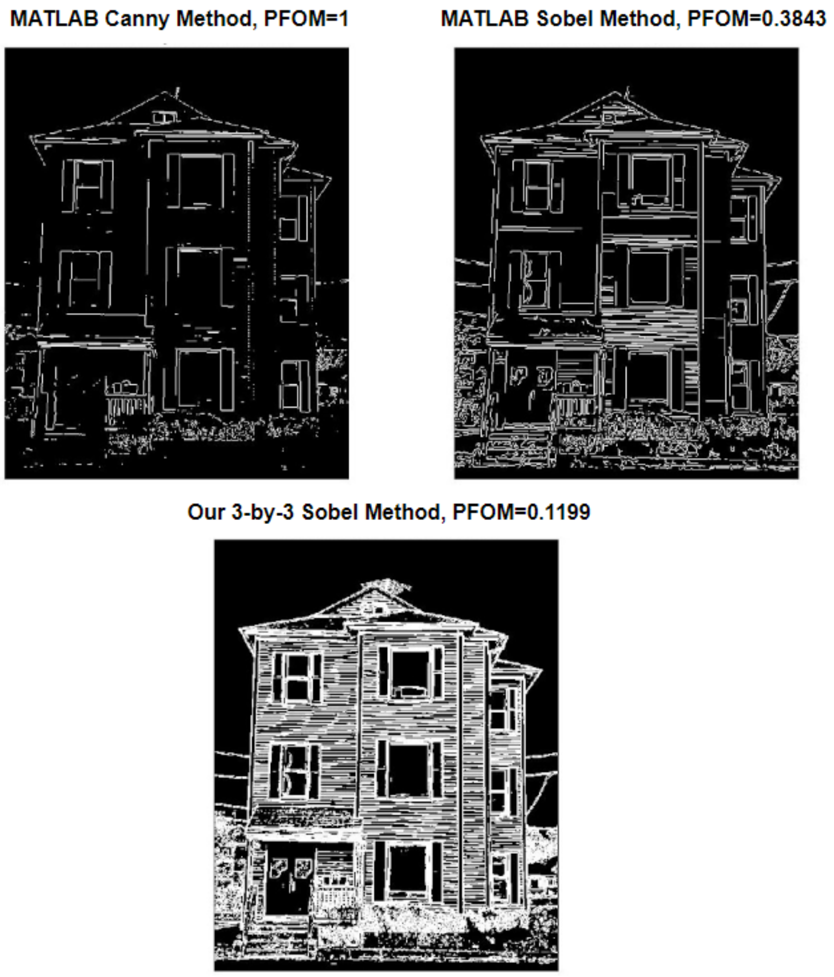
MATLAB Canny Method, PFOM=1
MATLAB Sobel Method, PFOM=0.3843
Our 3-by-3 Sobel Method, PFOM=0.1199

Figure 2.6: This figure demonstrates the Pratt Figure of Merit (PFOM) used to evaluate three images against the MATLAB automatic Canny method, via the pratt.m function created by Vivek Bhadouria and shared on the MATLAB Central File Exchange [34]. By row from left to right: the Canny Method, the MATLAB Sobel Method, and our 3-by-3 Sobel Method. As expected, we can see when comparing an image to itself, as done in the first case presented, the PFOM is 1, a perfect score. Then The MATLAB Sobel method yields a PFOM of approximately 0.3834 and our 3-by-3 Sobel method yields a lower value for the PFOM, only 0.1199. Visually, these scores make sense, as our method results in more false positive edges when compared to the Canny method, than the MATLAB Sobel method.

of success. A neural network is a machine learning algorithm which can be applied to both classification and regression problems [35].

Neural networks are considered deep learning methods, since they consist of a series of layers [35]. The basic structure of a neural network is an input layer, some number of hidden layers, and an output layer. The input layer corresponds with the size of the input data; in the case of an image classification problem, that is the number of pixels in the image [36]. The user's desired response dictates the size of the output layer, and in the case of classification, that is typically the number of categories in the problem [37]. Some of the layers can also be specialized to perform specific functions, such as how convolutional layers are often used in image processing problems [38]. One of the things that makes neural networks different from other machine learning models is that a neural network is tuned through a recursive training process which tunes the weights and biases of the network in order to refine the prediction algorithm and ultimately improve prediction accuracy [35].

Like all supervised machine learning methods, neural networks can produce a quantitative measure of accuracy, and in the case of classification, that accuracy is the proportion of correctly classified samples. Throughout the training process, the validation accuracy and training accuracy are used to inform the recursive training process, and a final testing accuracy should be calculated, on separate data not used for training or verification, to certify the network's performance [39]. Consistency between lower validation and testing errors indicate a well trained model, whereas a high error or significant discrepancy in these error values can indicate flaws in the model, such as overfitting the model training data [39].

No ground truth is needed to derive a quantitative measure of neural network classification accuracy. Instead, only a labeled dataset is needed to complete this analysis, and these are often readily available given the increase in digital data in recent decades [40]. It is this classification accuracy value that we think could serve as an objective, quantitative measure of edged detection method performance, in lieu of a figure of merit. Thus, this paper explores the use of a simple neural network as a tool for measuring and comparing the effectiveness of different edge detection methods.

3. Methods and Data

This section will describe the methodologies for edge detection and deep learning methods implemented in this project in order to explore the effectiveness of a variety of edge detection methods in a quantitative manner. Instead of measuring quality of edge detection methods according to a comparative method, like the Pratt Figure of Merit, which requires a ground truth reference, we instead measure edge detection effectiveness quantitatively with respect to a simple neural network's accuracy for image classification problems.

3.1 Edge Detection Functions

First we coded and tested a variety of edge detection methods during the project. All the methods used the same basic algorithm, with varying filters, as described in Section 2.1.4. The algorithm for these methods is detailed in the pseudocode provided in Algorithm 1, as modified from the MATLAB sample code available on the Sobel Operator Wikipedia page [41].

The first method we tried used simple filters derived from the central derivative equation, and used the in the second example in Section 2.1.4. This method, like many others, uses two filters: one for the derivative in the x-direction and one for the derivative in the y-direction. The precise filters are:

$$G_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The next method chosen for testing was the classic 3-by-3 Sobel method [26], as detailed in Section 2.2.1. There are also two filters used for this method, again with one relating to the x-direction and one the y-direction. The exact filters are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Next we included a modified Sobel method, mentioned in Section 2.2.3, which used a 5-by-5 filter [26]. Like the traditional Sobel method described above, this method uses two filters as well, again one for highlighting changes in the x-direction and the other for changes in the y-directions. The filters

Algorithm 1 Edge Detection Method

```
Function intakes image A
Define at least one n-by-n filter
Identify size of image A, r rows and c columns
Create matrix of all zeros, Z, same size as A
For demonstration we will define two filters, $G_x$ and $G_y$
initialize a counter $a = 0$
initialize a running sum, total=0
for i=1 to r-(n-1) do
    for j=1 to c-(n-1) do
        Apply $G_x$ to specified location in A, outputs S1
        Apply $G_y$ to specified location in A, outputs S2
        $S = \sqrt{S1^2 + S2^2}$, a 2-norm
        Assign S to corresponding location in Z, (i, j)
        Add S to running sum, $total = total + S$
        Adjust counter, $a = a + 1$
    end for
end for
$threshold = total/a$, an average of gradient values
Round pixel values in Z up or down according to threshold:
if $Z_{(i,j)} >= threshold$ then
    $Z_{(i,j)} = 1$ (white)
else
    $Z_{(i,j)} = 0$ (black)
end if
Return edge detected image as a logical binary type
```

are:

$$T_x = \begin{bmatrix} 2 & 3 & 0 & -3 & -2 \\ 3 & 4 & 0 & -4 & -3 \\ 6 & 6 & 0 & -6 & -6 \\ 3 & 4 & 0 & -4 & -3 \\ 2 & 3 & 0 & -3 & -2 \end{bmatrix} \quad \text{and} \quad T_y = \begin{bmatrix} 2 & 3 & 6 & 3 & 2 \\ 3 & 4 & 6 & 4 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ -3 & -4 & -6 & -4 & -3 \\ -2 & -3 & -6 & -3 & -2 \end{bmatrix}$$

Then we also included a second modified Sobel method, this one which used 4 different 5-by-5 filters [26]. As mentioned in Section 2.2.3, this method was designed with the intention to pick up on changes in the x- and y-directions as well as those on the 45 degree diagonals of the image. The first two filters T_x and T_y are the same as in the previous method described, whereas T_{45} and T_{135} are similar, but based on rotating the filter T_y 45 and 135 degrees, respectively. As with the other methods, the outputs of each filter are combined using a 2-norm. The precise filters for this method are:

$$T_x = \begin{bmatrix} 2 & 3 & 0 & -3 & -2 \\ 3 & 4 & 0 & -4 & -3 \\ 6 & 6 & 0 & -6 & -6 \\ 3 & 4 & 0 & -4 & -3 \\ 2 & 3 & 0 & -3 & -2 \end{bmatrix} \qquad T_y = \begin{bmatrix} 2 & 3 & 6 & 3 & 2 \\ 3 & 4 & 6 & 4 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ -3 & -4 & -6 & -4 & -3 \\ -2 & -3 & -6 & -3 & -2 \end{bmatrix}$$

$$T_{45} = \begin{bmatrix} 0 & -2 & -3 & -2 & -6 \\ 2 & 0 & -4 & -6 & -2 \\ 3 & 4 & 0 & -4 & -3 \\ 2 & 6 & 4 & 0 & -2 \\ 6 & 2 & 3 & 2 & 0 \end{bmatrix} \quad \text{and} \quad T_{135} = \begin{bmatrix} -6 & -2 & -3 & -2 & 0 \\ -2 & -6 & -4 & 0 & 2 \\ -3 & -4 & 0 & 4 & 3 \\ -2 & 0 & 4 & 6 & 2 \\ 0 & 2 & 3 & 2 & 6 \end{bmatrix}$$

Additionally, we tested the Laplacian-of-Gauss (LOG) method [32]. This method uses only one filter, and as described in Section 2.2.3, the coefficients are chosen with the intention of adding a blurring effect at the same time as it detects edges [32]. This filter can be seen to give the strongest weight to pixels close to the center, and smaller weights towards the diagonals and edges. The exact coefficients for this 5-by-5 filter are presented below:

$$G = \begin{bmatrix} -2 & -4 & -4 & -4 & -2 \\ -4 & 0 & 8 & 0 & -4 \\ -4 & 8 & 24 & 8 & -4 \\ -4 & 0 & 8 & 0 & -4 \\ -2 & -4 & -4 & -4 & -2 \end{bmatrix}$$

The results of each of these methods applied to the same image is demonstrated in Figure 3.1. Key places to look to find differences between method results are corners of shapes and off diagonal edges. Figure 3.1 also demonstrates the Sobel and Canny edge detection methods using the edges.m function available via MATLAB's image processing toolbox [11]. These examples use a version of automatic thresholding as described in Section 2.1.5 which is much higher than our auto-generated threshold for this test image, described in the next Section 3.2.

3.2 Automatic Threshold Selection

In order to fairly compare this variety of methods and avoid the hassle of heuristically determining an appropriate threshold for each of these five methods, we developed a simple algorithm for automatically calculating a threshold for defining edges, which only requires an image input.

This method is based on a mean value cut off, where each of the filtered values are totaled throughout the convolution process, then divided by the final number of pixel values retained by the edge detection process. This method automatically handles changes in filter size and number, as well as varying ranges of possible values a filter creates. Thus, it was understood that this average threshold method was a fair way to compare different methods' effectiveness against one another. More detail of this precise algorithm's implementation in the function is provided in the pseudocode explanation of the edge detection process (see Algorithm 1).

Moreover, we explored the idea of allowing the user to specify a scale factor for adjusting the threshold according to the average value calculated in this model. This can provide a user the choice to fine tune a threshold, but can detract from the automatic aspect of the method. Figure 3.2 shows how varied, user defined, scale factors can adjust the threshold and in turn affect the final edges detected by a method, in this case the Sobel method. This scale factor method was only explored in a limited capacity.

3.3 Datasets

In order to train and test an image classification algorithm, we must first identify labeled datasets of images to use. We used the extremely popular

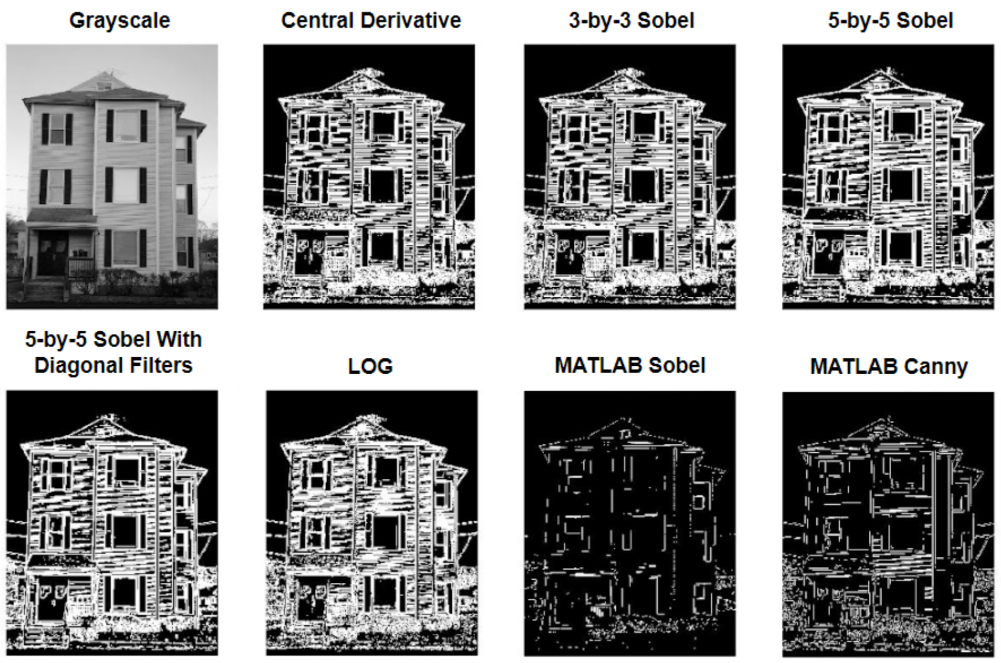

Figure 3.1: This figure illustrates the different methods detailed in this section, as applied to the grayscale image of a house. By row from left to right: the grayscale image of a house, the central difference method, the 3-by-3 Sobel method, the 5-by-5 Sobel method, the the 5-by-5 Sobel method with diagonal filters, the Laplacian-of-Gauss (LOG) method, the MATLAB Sobel method, and the MATLAB Canny Method. Key places to look to find differences are corners of shapes and off diagonal edges. The first five methods, excluding the grayscaled image provided for reference, use our automatic thresholding method, whereas the last two use the MATLAB auto-generated thresholding built into the edge.m function [11]

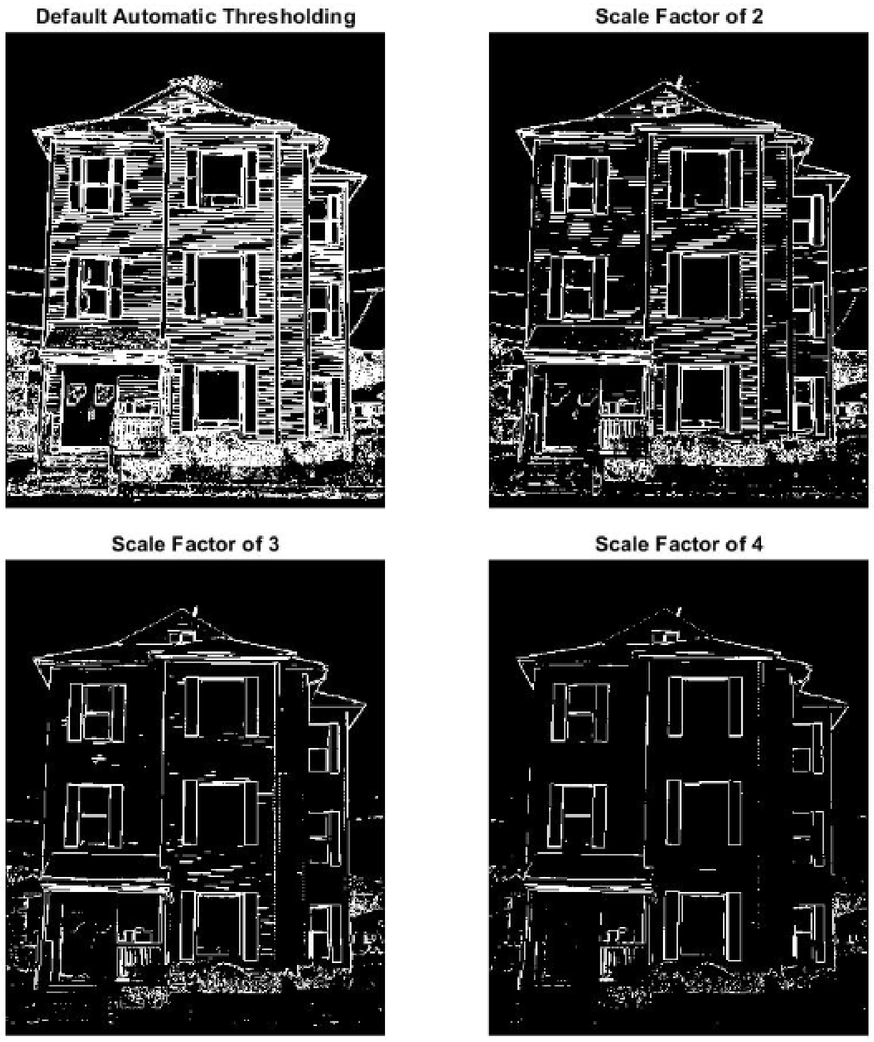

Figure 3.2: This figure illustrates how modifying the threshold for edge detection changes the output of the algorithm, according to a scale factor on the threshold. This example uses the Sobel method, initially with a scale factor of one, then subsequently larger thresholds, scaled by two, three, and four. By row from left to right: Sobel method with default threshold, Sobel method with threshold scaled by a factor of 2, Sobel method with threshold scaled by a factor of 3, and Sobel method with threshold scaled by a factor of 3. It is clear from the example that a low threshold maintains more information from the original image compared to the higher threshold, which erases more of the details in an image.

MNIST numbers dataset as a tool to set up and validate our network [42], then explored these edge detection methods on a more complicated dataset, the Architectural Heritage Elements dataset, which consisted of images of various architectural features [12]. Describing these datasets in detail is important, since the results are in the context of these particular datasets, which consist of a comparatively small number of relatively small images.

3.3.1 MNIST Dataset

The MNIST dataset consists of images of hand drawn numbers ranging from values 0-9 [43]. This dataset is automatically provided in the MATLAB deep learning package and contains 1000 images in each of the 10 categories [42]. Additionally, all the images are grayscale and consistently sized, only 28-by-28 pixels [42]. Together, these traits make the dataset a good tool for setting up a neural network and to ensure it is functioning as intended. This MNIST dataset is also used as the dataset in MATLAB's tutorial for building an image classification neural network [44]. We used this tutorial as a starting point when learning to build an image classification neural network in MATLAB, and returned to this dataset as a "sanity check" throughout the process working with the neural networks to ensure changes worked as expected and to aid in debugging.

The MNIST dataset itself is not a good candidate for edge detection experimentation though, since the majority of images are already simple lines and solid colored regions. Oftentimes, applying edge detection to these very small, simple images can actually make them more complicated opposed to simplifying the information, as intended. Figure 3.3 provides an example of how edge detection affects an image from the MNIST dataset.

3.3.2 Architectural Heritage Elements Dataset

In order to explore this method of using a simple neural network as a way of comparing and quantifying the effectiveness of edge detection methods, it was necessary to use a good image classification dataset for this type of experimentation. Some aspects taken into consideration when selecting a dataset were the size of each image and the total number of images available. Large images are more computationally expensive to both edge detect and use in machine learning. Similarly, if a dataset is very large, as in it contains many images, it will also be computationally expensive and slow to use for

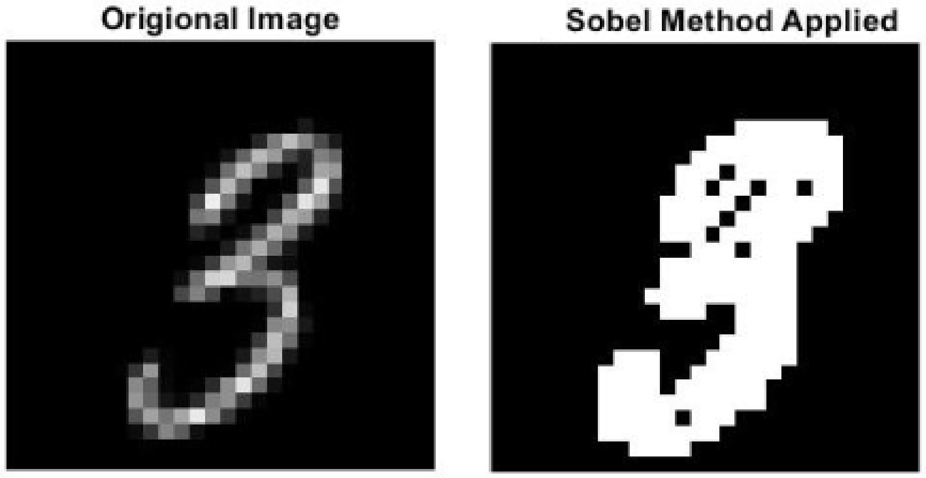

Figure 3.3: This figure illustrates an image from the MNIST dataset (left) compared to the same image with our 3-by-3 Sobel method algorithm applied (right) [43]. This comparison highlights that in some cases, such as this one where an image is very small and already represents a clear edge, an edge detection method can actually reduce clarity and add noise to a image, instead of improving it.

these applications. But alternately, if a dataset does not have enough images, there is little chance that a simple neural network will have adequate opportunity to learn and perform well, instead tending to over fit the training data [39].

Another consideration of the dataset would be the content of the images. Since many of the filters work in a combination of the x- and y-directions, we felt having at least some geometric features, such as those found in architecture, may be interesting and meaningful to include. Moreover, we wanted to choose a dataset where the images were high quality with a clear subject. Very noisy datasets tend to be difficult to work with for neural network training and testing [45], as well as edge detection [9], each with their own dedicated fields of study. Additionally, busy backgrounds can detract from the subject of an image, making classification problems more difficult.

With these factors in mind, we chose to use the Architectural Heritage

Elements (AHE) dataset for the experiments in this project, which had categories for a variety of architectural features like bell towers, vaults, and flying buttresses [12]. In total there were ten categories in the AHE dataset, and each category contained a different number of images. Each image is originally in color and of size 64-by-64 pixels. Figure 3.4 shows a selection of images, labeled with their category.

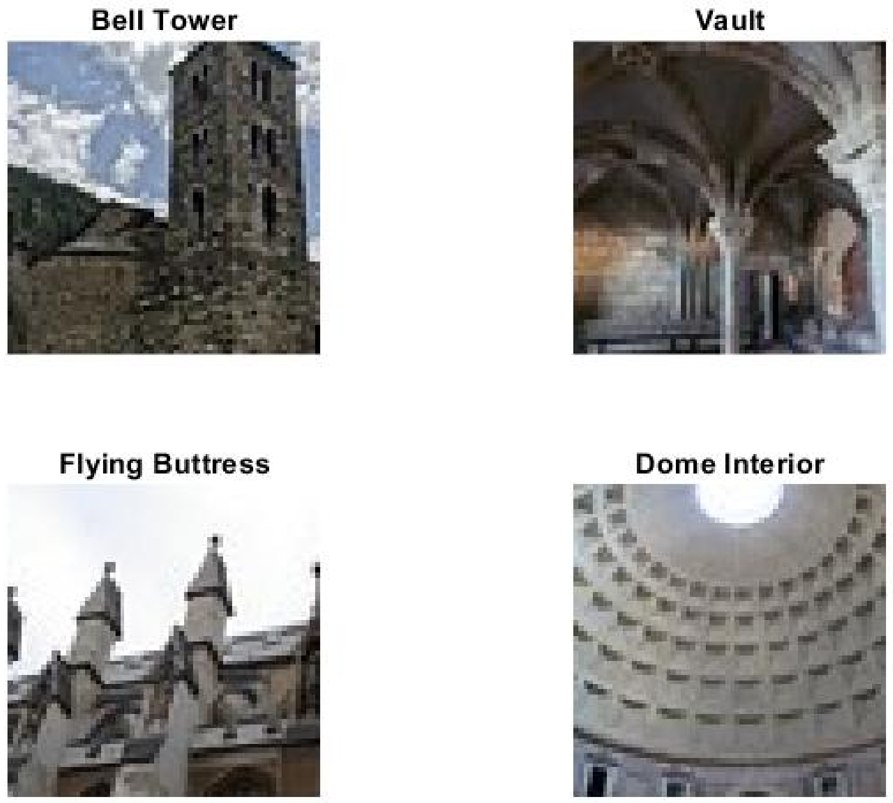

Figure 3.4: This figure exemplifies some of the original images from the Architectural Heritage Elements dataset [12]. The images are all 64-by-64 pixels in size and provided in color. By row from left to right: an image from the bell tower class, an image from the vault class, an image from the flying buttress class, and an image from the dome interior class.

3.3.3 Training, Testing, and Validation Sets

Instead of using all ten categories in the initial exploration of this problem, we instead created two smaller datasets from the ten categories provided. Reducing the number of categories simplified the classification problem, allowing us to focus more on the edge detection aspect of this project, opposed to tuning a complex neural network.

One of the datasets used consisted of only two of the larger, similarly sized categories, vaults and bell towers, with about 1000 images in each category's training set. The second dataset used consisted of three different categories (apses, flying buttresses, and dome interiors), with the number of images in each training category ranging from about 400 to about 600 images. We selected these three categories for the second dataset since they were the three most similarly sized groups. In order to apply edge detection to these images, they were all converted from color to grayscale.

The AHE dataset was accessed for this project through the website Kaggle, where the images were provided in folders according to their classification, and the data was pre-split into training and testing sets [12]. These groupings were used, as provided, in this project. For the first dataset, with only two categories, about 80-90% of the data in each category was used to train the neural network and 10-15% of the data was saved to do a final test of accuracy on the network, in keeping with best practices of cross validation [39]. The bell tower category had 1057 training samples and 170 testing samples, and the vault category had 1097 training samples and 163 testing samples. Within the training dataset, we also needed to determine which portion of the data would be a validation set, which is crucial to the recursive learning pattern of the neural network. To do this, the first 810 images from each category were used as training data for the algorithm, and the remaining images, 247 in the case of the bell tower group and 277 for the vault group, were used to validate the neural network during the training process. This equates to about 75% of the training data being used for training the neural network, and 25% of the training data being used for validation during the training process.

The second dataset, which was made up of three categories, contained fewer images per category and less even groupings compared to the previously described dataset. The second dataset was created with the intention of checking how this method would perform on a more challenging problem. Obviously adding a third group to the dataset makes the classification problem more challenging, and uneven datasets can also increase the complexity of a problem. For this reason this dataset would be used for experimentation with the edge detection methods as well. The apse category contained 505 training images and 50 testing images, the flying buttress category contained 405 training images and 70 testing images, and the interior dome category contained 589 training images and 69 testing images. Because of these unbalanced groups, there was not as clear a proportion of training to testing data within each category, but in total, based on these values, 89% of the data was used for training and the remaining 11% was used for testing. Again, in order to train the neural network via the back propagation method, we also needed to reserve a portion of the training data to be used as validation data, and based on the smallest category containing only 405 images, we decided to use the first 350 images from each category for training (in even amounts) and all remaining images in the training set were used for validation.

3.4 Neural Networks

For this project we used a simple neural network in MATLAB, created according to MATLAB's image classification deep learning tutorial [44]. This network was initially designed for the MNIST dataset and could achieve well over 90% accuracy on that data with very few epochs. With some modifications, the network was also suitable for AHE dataset. It was important to keep this network simple, with only a few layers, as to maintain focus on exploring edge detection methods opposed to tuning a neural network, which is its own area of research [35].

The network has seven layers total, the first being the image input layer. This layer corresponds with the size of the data, in the case of the AHE dataset, 64x64x1, for a 64-by-64 pixel, grayscale image [36]. The second layer is a two-dimensional convolutional layer, which is a layer which condenses data according to an image's two dimensional structure, using a dot product as the filtering operation [38]. The third layer of this network is a batch normalization layer, which is used to speed up training and decrease network sensitivity [46]. Next, the fourth layer is a Rectified Linear Unit (ReLU) layer, which applies a thresholding operation to the network to eliminate small, presumably negligible weights from the algorithm [47]. The fifth layer used in this network is a fully connected layer, which applies the weights and biases to modified input at this step [48]. Then the sixth layer in this network is a softmax layer, which gets its name from the softmax function that it applies to the input, preparing it for classification output [49]. The final, seventh layer in this neural network is the classification network, which uses cross-entropy to determine the final classification label for a datapoint according to the number of classes specified in the fully connected layer [37]. Based on this structure any correctly sized input can be fed to the network, and then the functions associated with each layer can be applied, one at a time, passing the output of one layer in as the input to the next until a final classification label is achieved, which is returned as the final output of the network.

Some other important parameters specified for this network were the learning rate, the validation frequency, and the number of epochs. The learning rate is the parameter that indicates the step size used in the gradient descent algorithm as part of the training process [50]. Using a step size that is too small can mean the network may take a very long time to train, whereas too large a step size can cause training to be inconsistent and sometimes lead to the network not learning at all [51]. Therefore it is important to pick an appropriate learning rate for a given problem, and through testing with this network, 0.01 appeared to work well for the methods tested in this project. The validation frequency is the measure of how often the network tests its accuracy on the validation data [51]. Finally, the number of epochs is defined by the number of times the network is presented with the set of training data. Thus the maximum number of epochs parameter indicates the total number of times the network will be presented with the training data points before terminating the training process [35].

4. Results

This section will discuss the results of the controlled experiments performed to explore the computational efficiency of a variety of the methods detailed in this paper and the potential use of neural networks accuracy as quantitative measures of edge detection methods in lieu of figures of merit. All results in this report were produced using a HP Pavilion Laptop with 16 GB RAM (15.9 GB usable), Intel® Core™ i7-8550U (CPU @ 1.80GHz), 4 cores, and using the MATLAB R2021b programming language.

4.1 Computational Efficiency

First we measured the computational efficiency of the various methods we described in Section 3.1. To test the methods we used two images from the bell tower and vault categories of the Architectural Heritage Elements dataset, selected at random, along with the house image used throughout the paper. The bell tower and vault image are each 64-by-64 pixels, and the house image is a much larger 569-by-713 pixels. Each function was tested three times on each of the selected images, so the results could benefit from replication, and to avoid any outliers skewing the data. The results of this experiment are provided in Table 4.1.

To illustrate these methods applied to an image from the AHE dataset, the bell tower image used for testing is shown in Figure 4.1, compared with all the processed versions of the image.

Method	**Avg Time Bell Tower Image (sec)**	**Avg Time Vault Image (sec)**	**Avg Time House Image (sec)**
Grayscale	0.00358	0.00348	0.02037
Gaussian Blur	0.00557	0.00545	0.04437
Histogram Equalization	0.00739	0.00583	0.03977
Central Derivative	0.01594	0.01485	0.97503
3-by-3 Sobel	0.01334	0.01429	0.97967
5-by-5 Sobel	0.01280	0.01299	0.99638
5-by-5 Sobel with Diagonals	0.02289	0.02231	1.97348
LOG	0.00824	0.00831	0.50155
MATLAB Sobel	0.00314	0.00340	0.01550
MATLAB Canny	0.00544	0.00571	0.03470

Table 4.1: Results of controlled experiments of timing edge detection methods. The average times are calculated from three replicates of the experiments and truncated at the fifth decimal place. Values reported are the average time needed to process the images of the bell tower, the vault, and the house, all reported in seconds. The bell tower and vault images are both 64-by-46 pixels and the house image is 569-by-713 pixels.

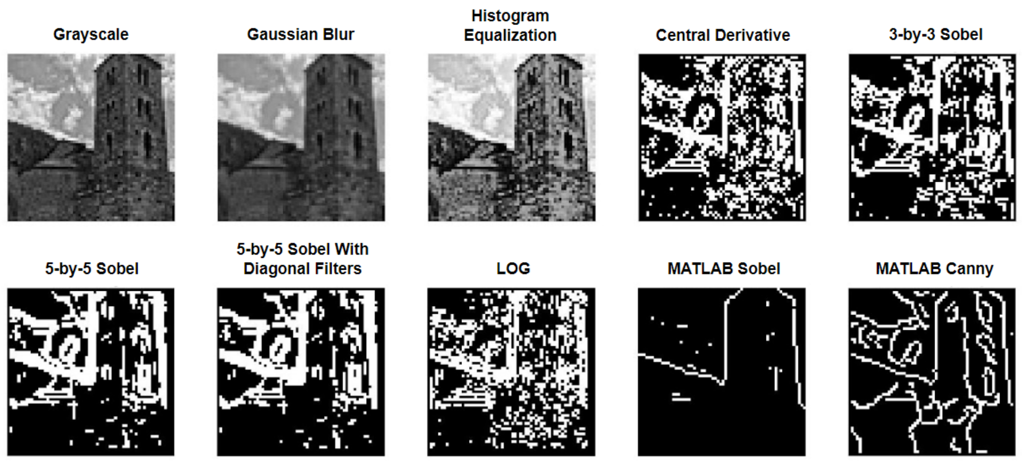

Figure 4.1: This figure illustrates the variety of methods applied to one of the images from the bell tower category of the Architectural Heritage Elements dataset [12]. By row from left to right: the grayscale image, the Gaussian blurred image, the histogram equalized image, the central derivative method applied to the image, our 3-by-3 Sobel method applied to the image, the 5-by-5 Sobel method applied to the image, the 5-by-5 Sobel method with diagonal filters applied to the image, the Laplacian-of-Gauss (LOG) method applied to the image, the MATLAB Sobel method applied to the image, and the MATLAB Canny method applied to the image.

4.2 Neural Networks

As for the neural networks, we measured and reported on the training time and accuracy values to analyze the effectiveness of the different edge detection methods in the context of deep learning. In order to be confident in the results, the training process and testing process for each network was replicated three times, in order to avoid any outliers skewing the results. This is especially important since the neural network training process involves randomness, so there can be significant variation in network training and performance, even on the same data. The average values of these three replicates for each response are reported in Tables 4.2 and 4.3. We report both the training accuracy and testing accuracy, as discrepancies in these values can indicate information on the true performance of the network, and the standard deviation of test accuracy, as an indication on consistency and repeatability of methods.

4.2.1 Two Category Classification

First we report the results in Table 4.2 for the neural networks designed to classify two categories of AHE data; bell towers and vaults. The method that yielded the best results in this test was the Laplacian-of-Gauss (LOG) edge detection method, without equalizing the image before application.

Method	Avg Validation Accuracy	Avg Testing Accuracy	Standard Deviation of Testing Accuracy	Avg Training Time (sec)
Grayscale	0.715	0.731	0.150	171.666
Gaussian Blur	0.667	0.642	0.132	172.666
Hitogram Equalization	0.672	0.729	0.190	184.333
Central Derivative	0.752	0.861	0.019	188.333
3-by-3 Sobel	0.775	0.889	0.054	187.000
5-by-5 Sobel	0.661	0.747	0.212	171.333
5-by-5 Sobel with Diagonals	0.649	0.730	0.190	169.333
LOG	0.842	0.924	0.019	176.333
MATLAB Sobel	0.697	0.828	0.012	186.333
MATLAB Canny	0.812	0.889	0.053	199.000
3-by-3 Sobel with Blur	0.782	0.861	0.104	169.000
3-by-3 Sobel with Equalization	0.842	0.900	0.038	183.000
LOG with Equalization	0.815	0.885	0.083	174.000

Table 4.2: Results of controlled experiments of accuracy and training time neural networks using the variety of edge detection methods on the two category dataset. Values reported are the average validation accuracy, the average testing accuracy, the standard deviation of the testing accuracy, and the time needed to train the neural network (in seconds). Averages are calculated from three replicates of the experiments and truncated at the third decimal place.

4.2.2 Three Category Classification

Next we report the results in Table 4.3 for the neural networks designed to classify images in the more challenging problem, the three categories dataset of apses, flying buttresses, and interior domes. We tested fewer methods than in the previous problem, after identifying the ones with the most promising results from the initial experiment. This experiment showed that many edge detection methods resulted in only marginally better accuracy compared to the grayscale image, with the only obvious improvement being attributed to the Canny method.

Method	**Avg Validation Accuracy**	**Avg Testing Accuracy**	**Standard Deviation of Testing Accuracy**	**Avg Training Time (sec)**
Grayscale	0.629	0.626	0.070	115.333
Sobel	0.602	0.652	0.026	123.333
LOG	0.524	0.601	0.102	119.333
MATLAB Sobel	0.492	0.536	0.058	113.666
MATLAB Canny	0.712	0.717	0.049	117.000
3-by-3 Sobel with Blur	0.666	0.696	0.154	122.000
3-by-3 Sobel with Equalization	0.619	0.617	0.006	113.000
LOG with Equalization	0.626	0.636	0.058	117.333

Table 4.3: Results of controlled experiments of accuracy and training time of neural networks using the variety of edge detection methods on the three category dataset. Values reported are the average validation accuracy, the average testing accuracy, the standard deviation of the testing accuracy, and the time needed to train the neural network (in seconds). Averages are calculated from three replicates of the experiments and truncated at the third decimal place.

5. Discussion and Future Work

In this chapter we discuss the interpretation of the results and additional opportunities for continued research and unanswered questions related to this project.

5.1 Discussion

We will begin by discussing the results as they pertain to computational efficiency and neural network behavior. The first key takeaway is that additional filter applications seem to be the feature of an edge detection method that most affects the time needed to implement it on an image. The second major result is that some of the methods developed without blurring and with our automatic threshold method outperformed the Canny edge detection method when implemented according to the automatic threshold method in MATLAB's edges function. This was true for the training and testing accuracy when using the two class dataset. This allows us to infer that in the context of deep learning, the best practices of blurring and thin edges are not always critical to yield favorable results in simple problems. But in more complex application of the three class dataset, the only method to cause significant improvement compared to the grayscale baseline was the Canny method.

5.1.1 Computational Efficiency

According to the results provided in Table 4.1, the slowest of the algorithms was the 5-by-5 Sobel methods with diagonal filters. This result is understandable as this method uses the most filters of all the methods tested (four) opposed to most other methods using only two. Alternately, the fastest method was the LOG method, which also makes sense given, since it is the only method that uses only a single filter. Moreover, the x- and y- central derivative method, the 3-by-3 Sobel method, and the 5-by-5 Sobel methods all averaged to similar amounts of time. We expected the 5-by-5 method to take the longest of these three methods in all scenarios, since it has the most complicated filter, followed by the 3-by-3 Sobel, then the very simple x- and y- central derivative method, but in fact the 5-by-5 methods were was actually faster in the case of small images. This could be because a 5-by-5 filter needs to be applied to an image fewer times, so in the case of small images, this advantage is noticeable, but in the case of large images, the additional calculations per filter application ultimately take more time.

None of our methods rivaled the times of the default functions in MATLAB. These functions include many additional steps beyond just detecting edges including blurring and edge reduction [11]. Additionally, the Canny method involves double edge detection and hysteresis steps. The default MATLAB functions are likely more computationally effective due to more efficient code structure, and therefore cannot be fairly compared to our more basic functions. Instead, the MATLAB functions are interesting to compare to one another, as it is clear that the MATLAB Sobel edge detection method is much faster than the MATLAB Canny edge detection method. In the case of the smaller images, the Canny method takes just less than double time compared to the Sobel method. But in the case of the larger image the Canny method requires a bit more than double the time. This can indicate how on a large scale it could save significant time to use the Sobel method over the Canny method, especially as image size increases.

Lastly, we notice that the preprocessing steps of grayscale, Gaussian blur, and histogram equalization all take significantly less time than any of our edge detection methods. This could indicate that if either of these optional methods result in significant improvement to network performance, they will contribute significantly less to computational time compared to the edge detection methods, so even a small improvement in performance could be worth the trade-off with time.

5.1.2 Neural Networks

From our data we first inspect the computation time to see there are only differences of a few seconds in the training time for the different networks, in both the two and three category datasets. We thought it could be possible that the datasets which had edge detection methods applied, resulting in binary logical elements could result faster training times compared to the grayscale images, since logical elements have pixel values of strictly 0 or 1 whereas grayscale pixel values range from 0 to 255. But in practice this was not the case, and we instead find that reducing the density of the input data via edge detection did not have any noticeable effect on training time in our experiment.

Next, we compare the average validation testing accuracy for the networks trained on all the methods. We can see that for both the two and three category datasets, the testing accuracy for all the networks is very close to or greater than their validation accuracy. This result is good, since if the testing accuracy were much lower for any of the models, this could be an indication of overfitting.

Now we consider the outcomes specific to the two category dataset, with results presented in Table 4.2. One of the first interesting results to discuss from this two category round of experiments is that some of the methods performed very similarly or worse than the networks trained on the grayscale images. This is true of the other reprocessing methods, where only blurring or histogram equalization was applied to the images, without applying an edge detection method. Both of these models performed worse on average than the grayscale model. It is also true that the 5-by-5 Sobel method with and without diagonal filters also yielded poorly performing models, resulting in worse validation and testing accuracy than the baseline of simple grayscale images. It is possible that this is due to the relatively small size of the images, such that the larger 5-by-5 filters could have combined too much information from far apart in the image into a single derivative, resulting in inaccurate edge detection. The major exception to this pattern is that the LOG operator performed the best out of any of the models tested on the two category dataset, and performed almost identically in testing when blurring was applied as a preprocessing step as to when it was not. It is possible that the heavy weighting of coefficient values towards the center of the LOG filter could have mitigated the problem that was seen when using either the 5-by-5 Sobel methods when on small images. Because of these initial results, we continued to test the LOG operator on the more challenging, three category dataset, but omitted both the 5-by-5 Sobel methods from additional testing.

Based on the experimental results for the two class dataset, we can also see that comparing the Sobel method with and without blurring, we achieved very similar results. The validation accuracy was slightly better for the method with blurring, whereas the testing accuracy was higher for the method without blurring. This indicates that blurring may not be a critical step when using edge detection in the context of simple image identification neural networks. In fact, since the method without blurring performed better on new data, it could indicate it was less likely to "memorize" the data, but instead uncover a more meaningful underlying pattern in the data, which it used for classifying the images. This discovery could relate to the way noise has been found to help improve training accuracy and develop more robust deep learning methods [20]. We can also see that in the case of the 3-by-3 Sobel method, applying histogram equalization as a preprocessing step prior to edge detection, and in lieu of blurring, yielded a significant improvement in both validation and testing accuracy for this network. This result could indicate that equalizing is a useful alternative to blurring in some contexts, but we cannot argue all, as these results were not supported by the LOG method accuracy values, which performed slightly better without histogram equalization applied prior to edge detection.

As for comments on threshold selection according to the two category dataset networks, we can see by comparing the automatic thresholding algorithms of the built in Sobel edge detection method in MATLAB's image processing toolbox [11] and our average-based method, our method had on average better results. As discussed in Section 2.1.5, the MATLAB method tends to set a higher threshold than our method, resulting in fewer false positive edges, but more false negative edges. On the other hand, the results from the MATLAB edge function, which has thinner edges, did yield more consistent results than our method, which has a standard deviation for testing accuracy that was almost five times the size of the MATLAB method. These results may indicate an underlying truth to the question regarding the trade-off between type I and type II errors in the context of edge detection for applications in deep learning. In context of the way a neural network works, with making linear combinations of the given inputs, it makes sense that false negatives could be dangerous, as they erase information that the deep learning model cannot bring back through its algorithm. But it is clear that too many false positives could lead to noise which causes inconsistent results in network training, as shown by the high standard deviation for our 3-by-3 Sobel method's testing accuracy.

Next we discuss the additional results from the second round of experiments, which used the three category dataset, as presented in Table 4.3. This experiment showed that the Canny Method and our 3-by-3 Sobel method with Gaussian blurring applied were the only methods to result in a noticeable improvement to prediction accuracy compared to the networks train on the baseline, grayscale images. In fact, all the other methods performed extremely similarly or worse than the neural networks trained on the grayscale images. These results could indicate in the context of more difficult problems that blurring can be a meaningful step in improving edge detection for image identification using deep learning, as opposed to using an alternate preprocessing step such as histogram equalization, or skipping the step all together.

One other result to note in relation to our questions is that the high threshold method produced by the MATLAB edge function for the Sobel method [11] performed significantly worse than our automatic thresholding method used with the 3-by-3 Sobel edge detection method. Without blurring first, our method which typically identifies a lower threshold compared to the automatic MATLAB function, and in turn retains more information, when used alone performed similarly to the grayscale image, and when used after Gaussian blurring, performed notably better that the baseline-grayscale image. This means, across both experiments, it was found that using a higher threshold and erring on the side of false-positives yielded better results compared to the lower thresholds that had greater potential to produce false negatives.

Ultimately, these experimental results and their analysis reveal that neural networks are absolutely a method for determining a quantitative evaluation of edge detection methods, though the results do not necessarily align with traditional metrics for validation. One drawback in using this method is that network tuning can be a time intensive process, especially for complicated datasets. Additionally, when a network is trained multiple times on the same data, the model will not always yield the same accuracy or training time results, meaning multiple tests will usually be necessary to yield meaningful results from this method.

5.2 Future Work

Based on this initial exploration of quantitatively evaluating edge detection methods using a simple neural network, there are many additional questions that can be explored. The first area to continue exploring next could be investigating these methods on new, alternative datasets. We briefly investigated datasets with noisy backgrounds and found limited success, so the question of noise would be of particular interest. Additional research could also be done for datasets of various size, or even color images, and it is very possible that the results of which edge detection methods perform best may differ. Looking into the behavior of neural network accuracy on any of these specific types of data could yield interesting results.

Moreover, additional filters and/or prepossessing steps could be explored in future work on this subject. In particular, it's possible that combining both Gaussian blurring and histogram equalization may be a beneficial step in highlighting important features in digital images, especially given the minimal computational cost of both these methods.

Lastly, additional work could be devoted to improving the automatic threshold method discussed in this paper, which was created to be easily applicable to many methods for the purpose of fair comparison. In particular, there may be the opportunity to collect additional information for an image, such as standard deviation of pixel differences, median pixel value, or image size, and use these values as a meaningful, automatic scale factor or additional term used to improve upon the average-based automatic threshold method we presented.

6. Conclusion

This project explored the possibility of implementing deep learning as a method of quantitatively comparing and evaluating edge detection methods in lieu of methods that require a ground truth comparison, like the Pratt Figure of Merit. In order to ensure fair comparisons, we implemented an average-based threshold selection method that was compatible with the various edge detection methods investigated in this work. We tested these methods on a simpler subset and more complex subset of the Architectural Heritage Elements dataset in order to investigate questions of interest regarding edge detection in the context of simple image classification problems using neural network solutions. Ultimately, this method allowed us to obtain quantitative measures of edge detection performance without a ground truth reference, with the major drawback being that results of this method are not consistent, and instead require repetition of experiments to yield meaningful results, due to the inherent randomization in the neural network training process.

In the second question we considered that, in the context of neural network validation on edge detection, the traditional step of Gaussian blurring might be eliminated while still maintaining good accuracy. Based on the experiments performed in comparing the Sobel method applied to the simpler, two category dataset, both with and without blurring, we achieved very similar accuracy results from the trained algorithms. Instead, we found histogram equalization to be a more effective preprocessing step compared to Gaussian blurring in this experiment. This indicates that blurring may not be a critical step when using edge detection in the context of a simple dataset, when attempting image identification using a neural network. But when datasets increased in complexity, such as in the case of the three category dataset with varying numbers of samples per category, this result was no longer found to be true, and instead Gaussian blurring did indicate improved neural network performance for image classification.

The last question regarded the idea of whether thinner edges are always preferable in the machine learning context. It was discussed that since type II errors can erase important information from the input, whereas the algorithm can still learn from and respond accordingly to thick edges, representative of type I errors, during the training process then it may be preferable in the context of deep learning to favor false-positives over false-negatives. To investigate this question we compared the automatic thresholding algorithm used in MATLAB's edge function and our automatic threshold method. In the context of the Sobel method, our threshold values for a given image were typically much lower, meaning more edges were maintained in the final edge map. In experiments using both the simple and more complicated datasets, we found the lower threshold to produce better average training and testing accuracy in the neural networks trained on that data. On the other hand, the results from the MATLAB edge function, which has thinner edges, did yield more consistent results than our method in the context of the simpler dataset, which may indicate that false-positives could also introduce potentially unwanted noise into the data.

Ultimately, this project explored many aspects of edge detection in the context of deep learning as a tool for quantitative evaluation of various methods. As edge detection has been a popular image processing tool for decades, we show with this work that there are still many opportunities to continue to experiment with these algorithms, especially thorough the lens of machine learning with neural networks.

Bibliography

[1] R. C. Gonzalez, "Digital image processing," eng, in *Digital Image Processing* (Applied mathematics and computation ; no. 13), Applied mathematics and computation ; no. 13. Reading, Mass: Addison-Wesley Pub. Co., Advanced Book Program, 1977, ch. Chapter 1 Introduction, ISBN: 0201025965.

[2] R. Jain, R. Kasturi, and B. Schunck, "Machine Vision," in Jan. 1995, ch. Introduction, ISBN: 978-0-07-032018-5.

[3] M. Sonka, V. Hlavac, and R. Boyle, "Image pre-processing," in *Image Processing, Analysis and Machine Vision*, Boston, MA: Springer US, 1993, pp. 56–111, ISBN: 978-1-4899-3216-7. DOI: `10.1007/978-1-4899-3216-7_4`. [Online]. Available: `https://doi.org/10.1007/978-1-4899-3216-7_4`.

[4] R. Jain, R. Kasturi, and B. Schunck, "Machine Vision," in Jan. 1995, ch. Chapter 5, Edge Detection, ISBN: 978-0-07-032018-5.

[5] I. Sobel, "An isotropic 3x3 image gradient operator," *Presentation at Stanford A.I. Project 1968*, Feb. 2014.

[6] MathWorks, *What Is Image Filtering in the Spatial Domain? - MATLAB & Simulink*. [Online]. Available: `https://www.mathworks.com/help/images/what-is-image-filtering-in-the-spatial-domain.html` (visited on 04/24/2023).

[7] R. Jain, R. Kasturi, and B. Schunck, "Machine Vision," in Jan. 1995, ch. Chapter 4, Image Filtering, ISBN: 978-0-07-032018-5.

[8] S. El-Khamy, M. Lotfy, and N. El-Yamany, "A modified fuzzy Sobel edge detector," in *Proceedings of the Seventeenth National Radio Science Conference. 17th NRSC'2000 (IEEE Cat. No.00EX396)*, Feb. 2000, pp. C32/1–C32/9. DOI: `10.1109/NRSC.2000.838961`.

[9] Z. Hameed and C. Wang, "Edge detection using histogram equalization and multi-filtering process," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, ISSN: 2158-1525, May 2011, pp. 1077–1080. DOI: `10.1109/ISCAS.2011.5937756`.

[10] R. C. Gonzalez, "Digital image processing," eng, in *Digital Image Processing* (Applied mathematics and computation ; no. 13), Applied mathematics and computation ; no. 13. Reading, Mass: Addison-Wesley Pub. Co., Advanced Book Program, 1977, ch. Chapter 4 Image Enhancement, ISBN: 0201025965.

[11] MathWorks, *Find edges in 2-D grayscale image - MATLAB edge.* [Online]. Available: https://www.mathworks.com/help/images/ref/edge.html (visited on 04/24/2023).

[12] I. Kobzev and V. Roman, *Architectural Heritage Elements Image64 Dataset*, en. [Online]. Available: https://www.kaggle.com/datasets/cf813b5fd4256b7ad7656efb0fb31b1c09686b0d8aa28c3c1f51f019e5203a77 (visited on 04/18/2023).

[13] MathWorks, *Deep Learning Toolbox Documentation.* [Online]. Available: https://www.mathworks.com/help/deeplearning/ (visited on 04/17/2023).

[14] G. S. Robinson, "Color Edge Detection," *Optical Engineering*, vol. 16, no. 5, p. 165 479, 1977. DOI: 10.1117/12.7972120. [Online]. Available: https://doi.org/10.1117/12.7972120.

[15] MathWorks, *Image types - MATLAB.* [Online]. Available: https://www.mathworks.com/help/matlab/creating_plots/image-types.html (visited on 04/30/2023).

[16] MathWorks, *Convert RGB image or colormap to grayscale - MATLAB rgb2gray.* [Online]. Available: https://www.mathworks.com/help/matlab/ref/rgb2gray.html (visited on 04/24/2023).

[17] MathWorks, *Convert matrix to grayscale image - MATLAB mat2gray.* [Online]. Available: https://www.mathworks.com/help/images/ref/mat2gray.html (visited on 04/24/2023).

[18] L. Shapiro and G. Stockman, "Computer Vision," in Prentice Hall, 2001, ch. Chapter 5 Filtering and Enhancing Images, ISBN: 978-0-13-030796-5. [Online]. Available: https://books.google.com/books?id=FftDAQAAIAAJ.

[19] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, pp. 679–698, Dec. 1986. DOI: 10.1109/TPAMI.1986.4767851.

[20] G. An, "The Effects of Adding Noise During Backpropagation Training on a Generalization Performance," *Neural Computation*, vol. 8, no. 3, pp. 643–674, Apr. 1996, Conference Name: Neural Computation, ISSN: 0899-7667. DOI: 10.1162/neco.1996.8.3.643.

[21] C. M. Bishop, “Training with Noise is Equivalent to Tikhonov Regularization,” *Neural Computation*, vol. 7, no. 1, pp. 108–116, Jan. 1995, ISSN: 0899-7667. DOI: `10.1162/neco.1995.7.1.108`. [Online]. Available: `https://doi.org/10.1162/neco.1995.7.1.108` (visited on 04/17/2023).

[22] MathWorks, *Adjust Image Contrast Using Histogram Equalization - MATLAB & Simulink*. [Online]. Available: `https://www.mathworks.com/help/images/histogram-equalization.html` (visited on 04/30/2023).

[23] F. Li, C. Shen, J. Fan, and C. Shen, “Image restoration combining a total variational filter and a fourth-order filter,” *Journal of Visual Communication and Image Representation*, vol. 18, no. 4, pp. 322–330, 2007, ISSN: 1047-3203. DOI: `https://doi.org/10.1016/j.jvcir.2007.04.005`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1047320307000260`.

[24] R. Jain, R. Kasturi, and B. Schunck, “Machine Vision,” in Jan. 1995, ch. Chapter 2, Binary Image Processing, ISBN: 978-0-07-032018-5.

[25] D. C. Montgomery, “Design and analysis of experiments,” eng, in 8th ed. Place of publication not identified: John Wiley & Sons Inc, 2013, ch. Chapter 2, Simple Comparative Experiments, ISBN: 1-62198-227-0.

[26] Z. Jin-Yu, C. Yan, and H. Xian-Xiang, “Edge detection of images based on improved Sobel operator and genetic algorithms,” in *2009 International Conference on Image Analysis and Signal Processing*, ISSN: 2156-0129, Apr. 2009, pp. 31–35. DOI: `10.1109/IASP.2009.5054605`.

[27] P. Vinista and M. M. Joe, “A novel modified sobel algorithm for better edge detection of various images,” *International journal of emerging technologies in engineering research (IJETER)*, vol. 7, no. 3, pp. 25–31, 2019.

[28] H. Kekre and S. Gharge, “Image segmentation using extended edge operator for mammographic images,” *International Journal on Computer Science and Engineering*, vol. 2, Jul. 2010.

[29] V. Mohan, “Performance analysis of canny and sobel edge detection algorithms in image mining,” *International Journal of Innovative Research in Computer and Communication Engineering*, pp. 1760–1767, Oct. 2013.

[30] A. Fuentes Alventosa, J. Gómez-Luna, and R. Medina-Carnicer, "Gud-canny: A real-time gpu-based unsupervised and distributed canny edge detector," *Journal of Real-Time Image Processing*, vol. 19, pp. 1–15, Jun. 2022. DOI: `10.1007/s11554-022-01208-0`.

[31] T. El Arwadi and A. El-Zaart, "A Novel 5x5 Edge Detection Operator for Blood Vessel Images," *British Journal of Applied Science & Technology*, vol. 11, pp. 1–10, Aug. 2015. DOI: `10.9734/BJAST/2015/19967`.

[32] Y.-y. Zheng, J.-l. Rao, and L. Wu, "Edge detection methods in digital image processing," in *2010 5th International Conference on Computer Science & Education*, Aug. 2010, pp. 471–473. DOI: `10.1109/ICCSE.2010.5593576`.

[33] Y. Yitzhaky and E. Peli, "A method for objective edge detection evaluation and detector parameter selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 1027–1033, Aug. 2003, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: `10.1109/TPAMI.2003.1217608`.

[34] V. Bhadouria, *Pratt's figure of merit*, 2023. [Online]. Available: `https://www.mathworks.com/matlabcentral/fileexchange/60473-pratt-s-figure-of-merit` (visited on 04/25/2023).

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[36] MathWorks, *Image input layer - MATLAB*. [Online]. Available: `https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.imageinputlayer.html` (visited on 04/18/2023).

[37] MathWorks, *Classification output layer - MATLAB classificationLayer*. [Online]. Available: `https://www.mathworks.com/help/deeplearning/ref/classificationlayer.html` (visited on 04/18/2023).

[38] MathWorks, *2-d convolutional layer - MATLAB*. [Online]. Available: `https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.convolution2dlayer.html` (visited on 04/18/2023).

[39] G. James, Daniela Witten, Trevor Hastie, and Rob Tibshirani, *An Introduction to Statistical Learning*, 1st ed. Springer, 2017, ISBN: 978-1-4614-7138-7. [Online]. Available: `https://www.statlearning.com/`.

[40] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. DOI: 10.1126/science.aaa8415. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8415. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaa8415.

[41] *Sobel operator*, en, Page Version ID: 1134260670, Jan. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Sobel_operator&oldid=1134260670 (visited on 04/17/2023).

[42] MathWorks, *Data sets for deep learning*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/data-sets-for-deep-learning.html (visited on 04/24/2023).

[43] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[44] MathWorks, *Create Simple Image Classification Network - MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/help/deeplearning/gs/create-simple-deep-learning-classification-network.html (visited on 04/17/2023).

[45] M. Momeny, A. M. Latif, M. A. Sarram, R. Sheikhpour, and Y. D. Zhang, "A noise robust convolutional neural network for image classification," *Results in Engineering*, vol. 10, p. 100 225, 2021, ISSN: 2590-1230. DOI: https://doi.org/10.1016/j.rineng.2021.100225. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590123021000268.

[46] MathWorks, *Batch normalization layer - MATLAB*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.batchnormalizationlayer.html (visited on 04/18/2023).

[47] MathWorks, *Rectified Linear Unit (ReLU) layer - MATLAB*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.relulayer.html (visited on 04/18/2023).

[48] MathWorks, *Fully connected layer - MATLAB*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html (visited on 04/29/2023).

[49] MathWorks, *Softmax layer - MATLAB*. [Online]. Available: `https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.softmaxlayer.html` (visited on 04/18/2023).

[50] E. Tuba, N. Bačanin, I. Strumberger, and M. Tuba, "Convolutional Neural Networks Hyperparameters Tuning," in *Artificial Intelligence: Theory and Applications*, E. Pap, Ed., Cham: Springer International Publishing, 2021, pp. 65–84, ISBN: 978-3-030-72711-6. DOI: `10.1007/978-3-030-72711-6_4`. [Online]. Available: `https://doi.org/10.1007/978-3-030-72711-6_4`.

[51] MathWorks, *Options for training deep learning neural network - MATLAB trainingOptions*. [Online]. Available: `https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html` (visited on 04/30/2023).