Gesture Classifier Using Convolutional Nerural Network

Adrian Ruvalcaba
College of Engineering, Virginia Tech
Blacksburg, VA 24060
adrianr@vt.edu

***Abstract*—This paper demonstrates a technique of classifying images of hand gestures into appropriate categories. The ability to recognize various different hand signals will be increasingly important as technology continues to advance by allowing effective communication without the need for talking. Applications of gesture recognition can be used in Sign Language interpretation to communicate with the disabled, in situations where there is a large amount of background noise such as concerts or large crowds, and even expanded to environments where sound does not effectively travel such as underwater or in outer space. By utilizing a convolutional neural network to create a classification algorithm, a high degree of accuracy in detection can be experienced. Additionally, neural networks give the ability to retrain with different datasets, allowing more gestures to be added without the need to change the core algorithm. This ability to modify and update gives the network a stability not possible with traditional "hard-coded" algorithms by allowing it to be used even when the original samples have been changed.**

1. *Introduction*

In this project, a Convolutional Neural Network was created and trained on a dataset of images of hand gestures. The desired results from the project would be a trained network that can efficiently and accurately return the value of the hand gesture provided. By using MATLAB, several scripts were created to apply both image manipulation algorithms as well as the creation and training of the convolutional neural network. The steps required to complete the program consisted of preparing the dataset, creating the convolutional neural network, and testing the trained network.

2. *Preparing the Dataset*

The convolutional neural network used as a classifier requires several different images of hand gestures to train and verify in order to achieve accurate results. As with all neural networks, the more data it has to train with, the better it performs [8]. The gathered images would be put through image manipulation filters before being sent to the neural network to achieve a cleaner result by eliminating imperfections in the image.

2.1 Creating the Dataset

In order to achieve accurate results, the program needed a large number of training images. These images would consist of hands displaying fingers relating to the digits zero to five. While more gestures can be added for detection, these six gestures are enough to show the functionality of the program.

I opted for creating my own dataset of images to train the neural network to assure accuracy and quality between images. A MATLAB script was created that would use a laptop webcam to capture images and save them to corresponding folders. Upon starting the database creation script, a live feed from the webcam is started. A bounding box is set up to indicate the location to hold your hand while performing the gesture captures. The program will capture ten images of each gesture before moving to the next one. These images will help the neural network account for small changes in images since each of the ten images will be slightly different based off natural movement of the user. The images are saved in their appropriate folders and the final image of each gesture is displayed as verification.

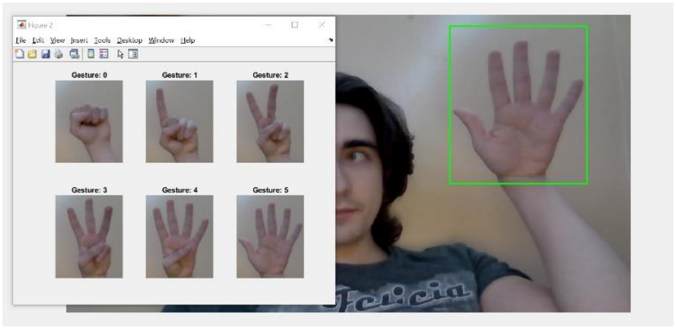


Figure 1: Database Creation

This program will be run as many times as the user requires to generate more images. By forcing the user to manually restart the program every ten images, it increases the probability of large changes in the images. That is, the user will hold up the gestures in a different location and orientation than the previous ten images, allowing more variety to the images and therefore increasing the effectiveness of the neural network.

The database creation program features the ability to change between left and right hands to add even more variability to the images. The bounding box location is moved to the opposite side and the filenames for the images are changed, but the procedure remains the same for image capturing. For the purposes of this paper and demonstration, 210 images were captured for each hand gesture on each hand, for a total of 2,520 custom training images.

2.2 *Image Manipulation*

Once the dataset was created, it was desirable to manipulate the images before sending them to the neural network. While the network could have taken the original images as inputs and began its calculations with the three-dimensional RGB images, applying custom filters to the image reduces the number of different features and increases efficiency while increasing speed. In addition, performing image manipulation would allow demonstration of different techniques learned throughout the course.

The aim of the manipulation step was to create an image that clearly displayed the digits in the hand while also removing unnecessary noise and features. Aspects of the picture such as shadows, fingernails, skin color, background, and imperfections in the hand can all contribute unnecessarily to the neural network as they don't impact the gesture being displayed, but they are still taken as part of the input image. To achieve this, three different techniques were attempted: Sobel Edge Detection, Sobel with Edge Direction, and simple threshold binarization.

The Sobel operator is a discrete differential operator mainly used for edge detection in images [11]. It consists of two sets of 3x3 matrices that detect horizontal and vertical differences in an image after convolution [9]. This method is prone to noise and can often heighten irregularities, so a gaussian blur was applied to the image before using the Sobel kernels. The following two matrices were used for Sobel [5]:

$$\underbrace{\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}}_{Gx} \qquad \underbrace{\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}_{Gy}$$

Once the horizontal and vertical features had been extracted, the size of the gradient was calculated using the following formula [5]:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

The direction of each edge can then also be calculated and used to further extract features from the image. The formula is as follows [5]:

$$\theta = \arctan(Gy/Gx)$$

Thresholding an image required a different technique for manipulating the image. Instead of convolving a kernel, the thresholding technique used involved the value of each individual pixel. If the value of the pixel was within a certain threshold, the corresponding pixel on an output image turns to white. If not within the threshold, it remains black. This technique, while simple, provides a binary image that can be less prone to background noise than the Sobel operator in this scenario. As with before, the image was turned to grayscale and a gaussian blur was applied before operating.

The results of the different image manipulation techniques can be seen in Figure 2.

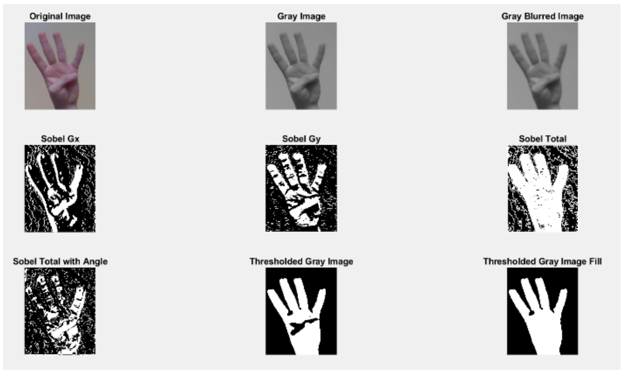


Figure 2: Image Manipulation Output

As observed, the Sobel operator did a fantastic job of detecting where the hand is, yet the image still had a large amount of noise. This was slightly reduced by modifying the direction of the angle of the Sobel output, but it was not enough to completely get rid of noise. Most likely this was caused mainly by the low-quality webcam used for database generation adding some background noise. This, however, was mitigated with the application of the thresholding function. The image displayed after the manipulation shows a very clear image of the hand and digits. It is easy to tell what gesture is being displayed and no other information is shown in the image. A fill was also added as further manipulation, resulting in a much cleaner image. This is the operation that will be performed on each individual image before being sent to the neural network to train. Figure 3 shows the output of the image manipulation script on several random input images.

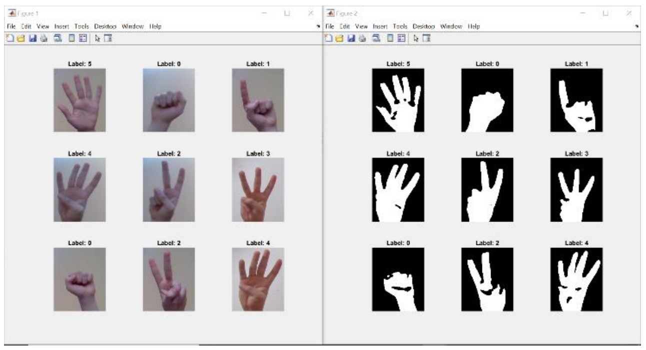


Figure 3: Output After Manipulation

3. *Convolutional Neural Network*

The convolutional neural network was responsible for getting the input images and learning how to classify them into the appropriate categories for hand gestures. This section will describe the process of creating the input layers, preparing the images, and training the network.

3.1 Layer Creation

Neural networks rely on a series of layers that perform computational operations on the input image to detect certain features unique to each image. These layers will be tuned as the program trains until it can accurately predict the hand gesture being shown. In convolutional neural networks, the layers consist of small kernels that perform convolutional operations on images [3]. These kernels allow for faster and precise calculations since it does not rely on each individual pixel. In addition, convolutional neural networks allow for accurate feature detection regardless of the location of the required gesture on the image [3]. This is helpful when gathering a large amount of outside sample data to train since it still be able to detect the hand gesture regardless of changes to the image.

3.1.1 Input Layer

The first layer in the network was the Image Input Layer. This layer serves the purpose of preparing the image for the remaining layers. The layer gets the size of the input image and performs data normalization to achieve a clean image for calculation. The sample images I generated for the database had a size of [382 312 1], or 382x312 pixels with a dimension of 1 for grayscale image.

3.1.2 Hidden Layers

The second layer was the Convolution Layer. This layer is responsible for performing operations on the input image based off weights and biases. The convolutional part of the layer means that a small kernel will be convolved with the image to produce the output instead of looking at the whole image. Below is a graphic from MachineLearningMastery [3] regarding the operation of the convolutional layer

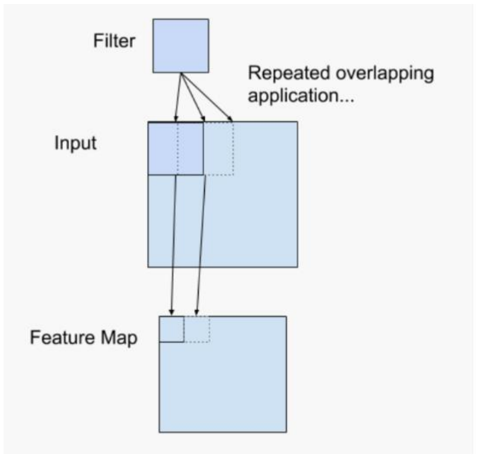


Figure 4: Convolution Layer

The kernel used in the convolutional layer will be a 3x3 filter. The kernel will have randomized weights [4] to begin with and it will output the resulting convolved image to an output node. The convolutional layer can have many different output nodes to increase prediction accuracy. Each of the nodes will have a different kernel that detects a different feature of the image. The kernels are automatically modified during the training of the network to achieve the desired results.

After the convolutional layer is processed, it is desirable to clean up the output data before allowing more layer operations. The purpose of this is to reduce the unnecessary features of the image that are not needed in the feature detection. Performing data normalization optimizes the neural network as it allows for less feature calculation down the line.

The first of these layers is the Batch Normalization Layer. This layer dramatically reduces the number of epochs (number of times the entire dataset is calculated) by standardizing the data [1]. This layer rescales the output of the Convolution Layer to have mean zero and a standard deviation of one [2]. This is effective during back propagation since it does not have to recalculate these values and instead can focus on updating kernel weights for the convolutional layer [8]. The algorithm for Batch Normalization is as follows where x_i are the inputs, μ_B is the calculated mean, and $\sigma^2{}_B$ is the variance.

$$\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}.$$

The batch gets further improved by applying the following equation, where γ and β are learnable parameters that get updated during the training of the network.

$$y_i = \gamma \widehat{x}_i + \beta.$$

After the Batch Normalization Layer, the image is sent through a Rectified Linear Unit Layer (ReLU). This layer performs a thresholding operation to convert negative numbers to zero. This removes some unwanted features that may have appeared during the Convolution Layer since prominent features will have positive values already and there is no need for the network to waste time calculating the negative values.

The ReLU layer performs the equivalent operation of the following function:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

Once the data has been normalized, a Max Pooling Layer is applied. This layer is used to traverse the output from the previous layer and return the maximum value of a 2x2 kernel. As with the other optimization layers, this serves to amplify the prominent features in the image while reducing overall training time [7]. Given the kernel used in the network, each Max Pooling layer reduces the image by a factor of 2, reducing the computational time of the following layers.

3.1.3 Output Layer

After the hidden layers, a Fully Connected Layer was created to begin image classification. This layer consists of six nodes, one for each of the possible hand gestures. Each of the nodes is connected to each node in the layer before it to gather all the feature data extracted through the hidden layers. As with the previous layers, the output nodes have a weight and a bias that is modified as the network trains in order to achieve accurate results.

Before classification, the Fully Connected Layer goes through a Soft Max Layer. This function is used to convert the values of the output nodes into a probability [10]. Each of the nodes will have a value between [0,1] and they will sum to 1. The equation for the Softmax function is as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The final layer in the network is the Classification Layer. This layer takes the probabilities of the previous function and returns the value corresponding to the hand gesture it has recognized throughout the layers.

3.1.4 Full Network

The Convolution Layer, Batch Normalization Layer, ReLU Layer, and the Max Pooling Layer are used together and repeated as many times as required to achieve accurate results. In the neural network used for this project, four of these hidden layer sets were used. The values for each of the layers were identical except for in the Convolutional Layer, where the number of output nodes was increased as the network progressed. This was done with the intent of increasing accuracy by allowing more features to be detected the deeper the image got into the network. The values of the output nodes were 8,16,32, and 64 nodes.

The structure of the Convolutional Neural Network can be visualized as seen below where the first layer is the input image, the final layer is the classification, and the middle sections are hidden layers.

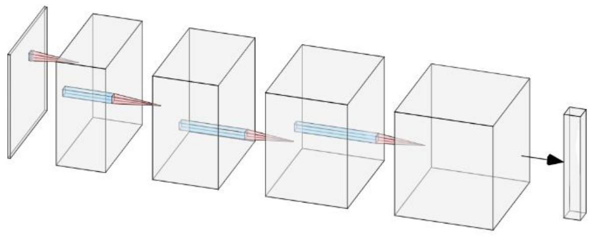
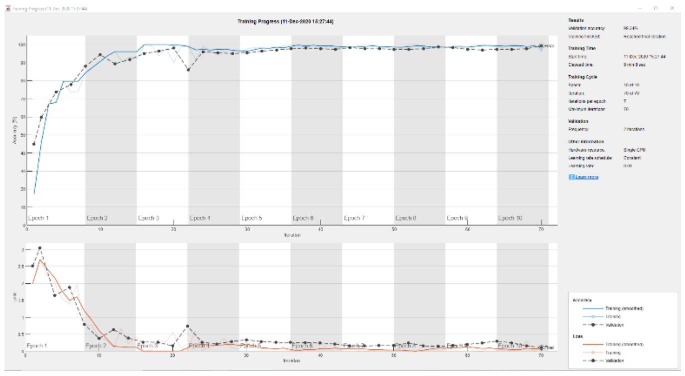


Figure 5: Visualized Convolutional Neural Network

3.2 Network Training

Once the network was created, training was needed before classification could be performed. Using the MATLAB network trainer along with the convolutional neural network created in the previous steps, the network took the input images and attempted to learn form them. The trainer would have a learn rate of 0.01 to modify the weights and biases, a max epoch of 10, and validate the results every two iterations. With these values, the network was able to achieve the desired results without the need for long calculation times. The training process plot is displayed below. The top graph displays the accuracy of the network, and the bottom graph displays the error.

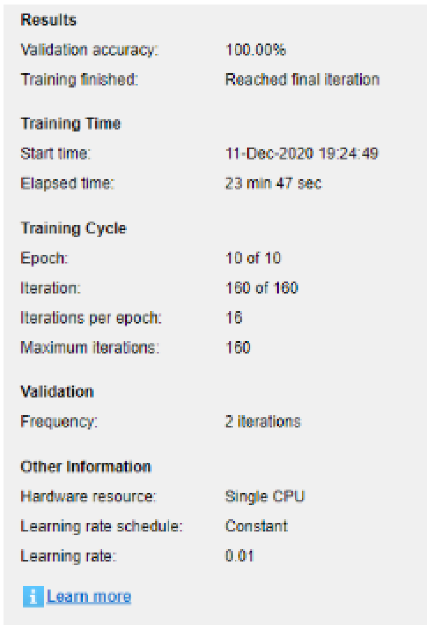


Results
Validation accuracy: [illegible]
Training finished: Reached final iteration
Training Time
Start time: 11-Dec-2020 15:27:44
Elapsed time: 9 min 3 sec
Training Cycle
Epoch: 10 of 10
Iteration: 70 of 70
Iterations per epoch: 7
Maximum iterations: 70
Validation
Frequency: 2 iterations
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.01

Figure 6: Network Training Output

4. *Experimental Results*

The convolutional neural network created in the project showed great success. After optimizing the network and training with the created dataset, the network was able to accurately classify images sent. With the hand gesture dataset I created, the network was able to achieve 100% validation accuracy, as shown below.



```
Results
Validation accuracy:          100.00%
Training finished:            Reached final iteration

Training Time
Start time:                   11-Dec-2020 19:24:49
Elapsed time:                 23 min 47 sec

Training Cycle
Epoch:                        10 of 10
Iteration:                    160 of 160
Iterations per epoch:         16
Maximum iterations:           160

Validation
Frequency:                    2 iterations

Other Information
Hardware resource:            Single CPU
Learning rate schedule:       Constant
Learning rate:                0.01

Learn more
```

Figure 7: Trained Network Results

Testing the program with random images from the dataset proved to provide accurate results. Below is a figure displaying nine random hand gestures along with a label representing the detected value.

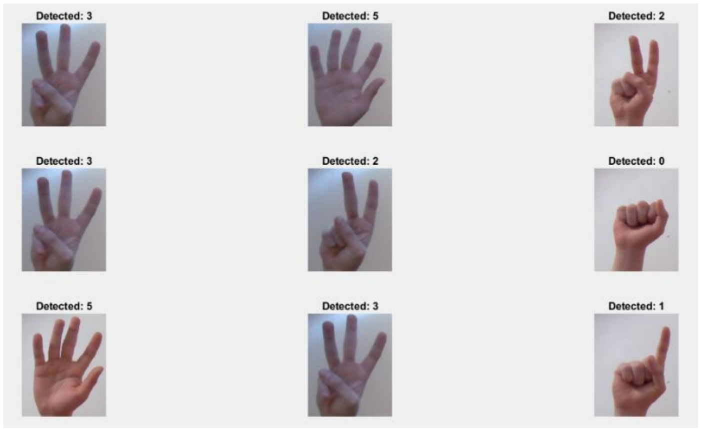
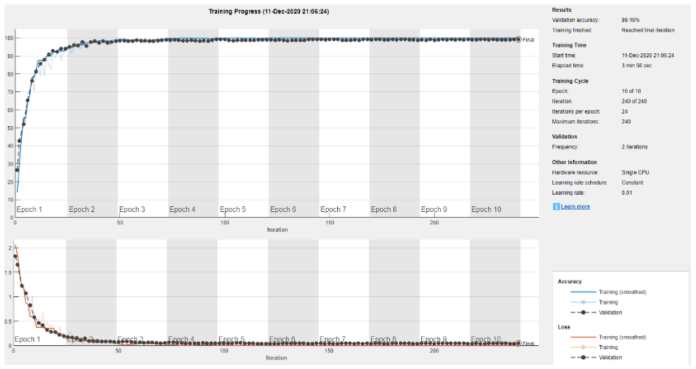


Figure 8: Trained Network Testing

Upon testing the network with images that are outside of my dataset, however, the network started making incorrect predictions. This is likely due to the fact that the images I created are very "clean" images with little imperfections and simple backgrounds that can be easily interpreted. To fix this issue, I simply added more images from the internet [6] to the initial dataset and retrained the data. This is a similar process to what would be done if more gestures are to be added. None of the code was modified, and the new network was then able to detect a much wider range of images. Below is the result of the training graph as well as the detection montage.



```
Results
Validation accuracy:          99.16%
Training finished:            Reached final iteration

Training Time
Start time:                   11-Dec-2020 21:06:24
Elapsed time:                 3 min 56 sec

Training Cycle
Epoch:                        10 of 10
Iteration:                    240 of 240
Iterations per epoch:         24
Maximum iterations:           240

Validation
Frequency:                    2 iterations

Other Information
Hardware resource:            Single CPU
Learning rate schedule:       Constant
Learning rate:                0.01

Learn more
```

Figure 9: New Network Training Output

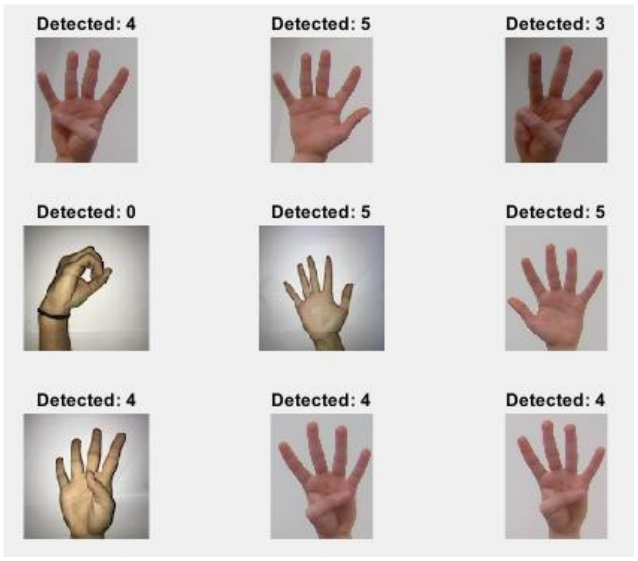


Figure 10: Trained Network Testing

Given the variability in the sample data, the accuracy of the validation was dropped from 100% to 99.16% in the newly trained network. While 100% would be optimal, given all the different factors with the differences in each gesture, the accuracy present is more than enough to accurately categorize the gestures. To further test the network, I asked a colleague to send me a picture of his hand using a mobile phone. Knowing the image would be far different than the images gathered online and the ones taken with my laptop camera, this was a good way to test if the network was accurately predicting the gestures. As expected, the network was able to classify the gesture correctly and is shown below:

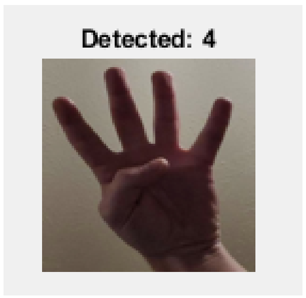


Figure 11: Colleague Testing Image

5. *Discussion*

Neural Networks are very important tools that can be used in a wide range of applications. As humans develop newer technologies, the amount of data that needs to be calculated increased dramatically. By utilizing neural networks to learn from the given data, we can achieve a high accuracy from the outputted results.

The scope of this project was to develop a convolutional neural network that can successfully interpret images of hand gestures into the intended meaning. As shown through the report, the method applied had a high degree of accuracy and can successfully categorize input images into the appropriate gestures. This method can be expanded to gather information on many different gestures not trained in this project, allowing it to be used in a wide range of applications.

Future goals for the project include adding a wider range of gestures to the recognition algorithm with the hopes of a full Sign Language Interpreter. Given enough data, neural networks such as this one can accurately learn to perform the tasks we give it with a degree of accuracy not possible with other programming methods.

6. *References*

1. Brownlee, Jason. “A Gentle Introduction to Batch Normalization for Deep Neural Networks.” *Machine Learning Mastery*, 3 Dec. 2019, machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.
2. Brownlee, Jason. “A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size.” *Machine Learning Mastery*, 19 Aug. 2019, machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.
3. Brownlee, Jason. “How Do Convolutional Layers Work in Deep Learning Neural Networks?” *Machine Learning Mastery*, 16 Apr. 2020, machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.
4. “How Are Weights Represented in a Convolution Neural Network?” *Data Science Stack Exchange*, 1 May 1966, datascience.stackexchange.com/questions/18341/how-are-weights-represented-in-a-convolution-neural-network.
5. Hypermedia Image Processing Reference. “Sobel Edge Detector.” *Feature Detectors - Sobel Edge Detector*, homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm.
6. Mavi, Arda. “Sign Language Digits Dataset.” *Kaggle*, 24 Dec. 2017, www.kaggle.com/ardamavi/sign-language-digits-dataset.
7. Ognjanovski, Gavril. “Build Hand Gesture Recognition from Scratch Using Neural Network - Machine Learning Easy and Fun.” *Medium*, Towards Data Science, 7 June 2020, towardsdatascience.com/build-hand-gesture-recognition-from-scratch-using-neural-network-machine-learning-easy-and-fun-d7652dd105af.
8. Ognjanovski, Gavril. “Everything You Need to Know about Neural Networks and Backpropagation - Machine Learning Made Easy...” *Towards Data Science*, Towards Data Science, 7 June 2020, towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a.
9. Tutorials Point. *Sobel Operator*, www.tutorialspoint.com/dip/sobel_operator.htm.
10. Wood, Thomas. “Softmax Function.” *What Is the Softmax Function?*, DeepAI, 17 May 2019, deepai.org/machine-learning-glossary-and-terms/softmax-layer.
11. Yuheng, Song, and Yan Hao. “Image Segmentation Algorithms Overview.” *ArXiv.org*, Cornell University, 7 July 2017, arxiv.org/abs/1707.02051.