***Abstract—* This paper focuses on comparing the effects of scaling an image on Convolutional Neural Networks (CNN). More specifically this paper is composed of the literary background on images, image classification, neural networks, and convolutional neural networks. Proposed in this paper is a primary research objective, goal tree, hypothesis, a milestone schedule, and an experiment design and solution description related to image classification.**

1. Introduction & Problem Description

Teaching machines how to see clearly, is one of many new exciting challenges in the field of machine learning. A discussion of one available the image classification methods is as follows in this paper and will be composed of three fundamental components: defining an image’s structure and subject (image classification), preparing images for view, training, and testing (image preprocessing), and utilizing machine learning tools and algorithms of deductive reasoning to determine what the object may specifically be (neural networks) [1]. As computers and artificial intelligence continue to evolve, its ability to ‘see’ is paramount, as new and creative ways employ and advance the partnership between man and machines.

1. Image Classification

A growing and evolving area in the field of machine learning is image classification, which is the complex process of assigning a predefined class label to an image based on its pixels. Familiar sophisticated examples of real world examples include implementation in autonomous vehicles, security cameras, quality control in manufacturing, social media and website content tailoring and monitoring, and satellite drone and drone analysis [1]. Exploring and improving existing classifiers, or the process of classification, could create new implementation opportunities, such as lessening human errors, and creating improvements in the fields of medicine, military defense, and security, to name just a few.

Computers can use artificial neural networks (described later) to perform image classification and to recognize or group images into any number of predefined classes. Classification, especially with neural networks, has the objective to filter out the higher level of importance data, such as edges and patterns from the images as a whole, into a resulting more finite matrix. From this matrix analysis may lend to faster, more accurate, and less cumbersome computations [1]. Classification of any neural network method strives for accuracy, but it requires a lot of training to self-improve [2].

There are a number of ways to classify images and popular techniques include neural networks, support vector machines, and k-nearest neighbors. In Daniel Nelson’s article, “*How Does Image Classification Work?*”, Nelson explains that neural networks are most commonly employed due to their stronger management of nonlinear relationships which may produce higher accuracy as a result. In contrast k-nearest neighbors (KNN), a type of learning algorithm, works well for small data sets, but can be handicapped by misclassification errors and slows down intolerably as the size of the datasets increase.The misclassification error possibly stems from calculating similarities based on all features equally [2]. This can be a problem when only a small subsection, or subset, of the feature elements from a provided image are highlighted for classification. Another method commonly used is support vector machines (SVM). They are also capable of nonlinear classification, but are crippled by large data sets, as they are limited in size and speed to perform computations.

However neural networks are not without faults. One of two major weaknesses is handling the presence of any non-convex loss functions [2]. The second issue is that neural net classifiers also depend on datasets with a vast number of images to train effectively and as such face steep computation times [3]. To combat or atone for these pitfalls, without sacrificing the benefits, a customized version of neural networks is typically used. This altered algorithm is called a Convolutional Neural Network (CNN), and this technique uses a filter to extract the most defining pixels of an image [2]. The advantage to this algorithm is the filters act as method feature extraction. The filtering process reduces the parameter through progression of the network, without losing the important data from the input. This approach allows for the model structure to decrease in complexity while the network learns in parallel [4]. To reiterate, CNN’s can be very costly to initially train but the benefits of accuracy, decreasing complexity in the structure traversal, and the ability of reverse traversal for training are worth the costs of longer computation times [2]. These areas will be discussed later throughout the paper.

1. Image Preprocessing

Prior to implementing a neural network model, , the input images must be prepared for machine learning. To start, an image is defined as a two-dimensional function, F(x,y), or a matrix with specifically arranged columns and rows [5]. Where x and y represent spatial coordinates, and each (x,y) pair is called an amplitude. If the elements within the matrix and the amplitudes of the (x,y) pairs are discrete and or finite, then the matrix is considered a digital image [6]. Similarly, if the matrix is considered a digital image, then the individual coordinates can be termed as pixels. Pixels are the smallest elements of images and hold a value, or an intensity. This intensity value is the most important piece in classifying images, as a pixel's intensity value can be represented as an integer from a large range of numbers to be analyzed. Intensity is commonly called the gray level and is often represented in an 8-bit integer [6]. This means the values can be any integer values between 0 and 255. Darker regions are represented as 0, whereas 255 indicates brighter regions [6].

The three main types of images are binary, grayscale, and color. Binary images consist of pixels that can be only one of two intensities or gray levels, 0 and 255. There are no in-between or intermediate values; the pixels can only be black or white. Grayscale images may include the range of intermediate values between 0 and 255. Color images, on the other hand, are the more complex of the three basic image types. Intensity values still apply however, to have color there needs to be three pixel values each representing a level of Red, Blue, and Green (RGB) [6]. In image classification, when analyzing images in a RGB color space, the classifier needs to not analyze one matrix, but instead three matrices that together comprise the digital image [7].

Building on function and image type, there are additional techniques and tools to consider when preparing image data for machine learning. The first, and arguably most important step in preparation is image scaling and making all the images uniform in size for training and testing. This will ensure some constant physical bounds within each test case, even if the contents are different [1]. Another decision to make is selecting the image type from the three mentioned previously (binary, grayscale and color). Image type will notably affect the Convolutional Neural Network’s extraction layer feature, as it will need to either pull from one matrix or three. Image type may also influence what kind of activation function will be used in the network. These concepts will be explored in more depth later in the paper, but selecting an image type may include a simple color conversion from RGB to grayscale, or vice-versa. Another aspect that is important to consider is the effect of image noise on classifier accuracy. Noise is simply a random variation in color or brightness. An overabundance in image noise is not ideal [1]. Therefore it is wise to reduce this factor, since noise comes from an external source and is not inherently a trait of the subject in the image, however this will not pose a large issue in the dataset that will be used in the experiment and solution descriptions [1].

1. Neural Networks

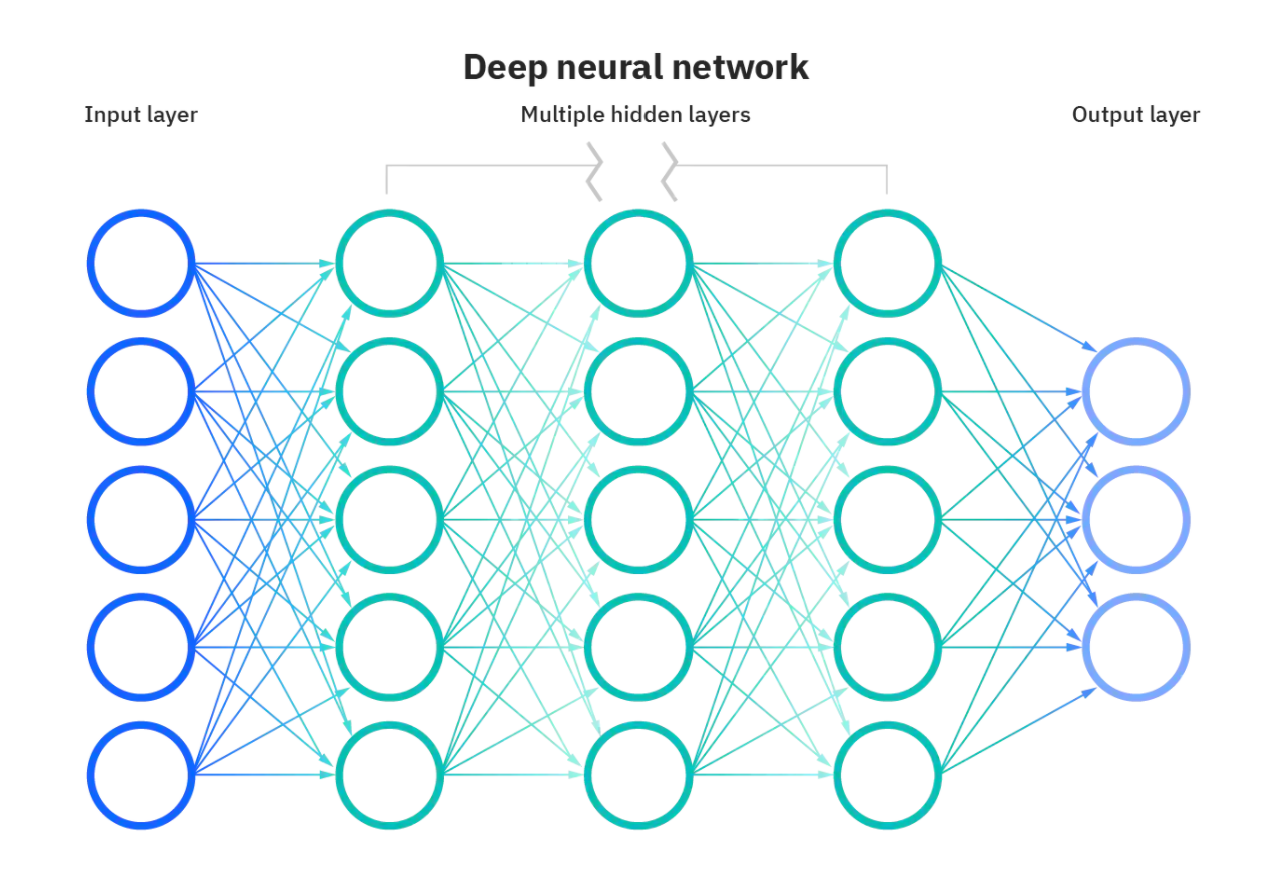
Neural networks are a machine learning technique that is tightly coupled with deep learning. The structure of these models are reflective of human neurons and simulates the behavior of the brain when processing decisions and recognizing images [7]. Sensory neurons are cells responsible for receiving sensory input, and invoking action by sending chemical and electrical signals to the rest of the body. These units are the influence for the nodes within a neural network. Nodes are the building blocks of a neural nets’ structure by forming connected layers. The first layer of nodes is typically called the input layer (see Figure 1). The following one or more layers are called hidden layers, which finally connect to a single output layer [7].

Figure 1 [7]

With the structure in mind, feed-foward propagation can be explained. IBM suggests that each node should be treated as its own linear regression equation [7], and to consider that it has two states [8]. The two states a hidden layer or output node can be in are pre-activated and activated. The pre-activated state is where the weighted sum of inputs is calculated, by taking the product of the input and the associated weight of the previous contributing nodes [8]. The activated state is where the weighted sum of the pre-activated state is passed into an activation function. An activation function is a function that is used to aid the network in learning complex patterns. Activation functions are also responsible for the important task of deciding if the neuron should be activated and if the input should be transformed and made to be an output [9]. There are two types of activation functions as well; linear and nonlinear [9]. Non-linear functions are most frequently used as they can be used for higher degrees of complexity unlike linear types. The most common activation functions are ReLu, Tanh, and Softmax. Sigmoid is well known but it is expensive and is usually just used for historical reasons and binary classifiers [9].

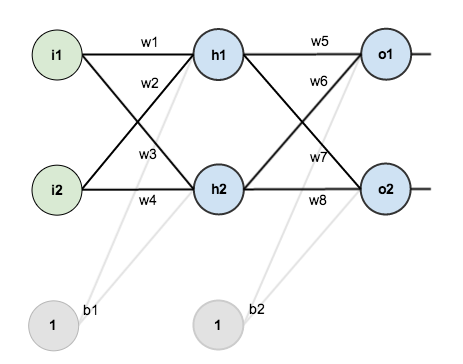


Figure 2 [10]

For the sake of simplicity, Figure 2 represents a small neural network of 3 layers. It consists of two input nodes (i), two hidden layer nodes (h), two outputs (o), and two biases (b). For simplicity treat all layers from Figure 2, as the variable X just for the in the equation below in Figure 3.Returning back to traversal through a node, the result of the activation function surpasses the threshold, then the node may pass the data to the next node [8]. This is like a biological neuron firing a signal to the next neuron, and passing a message. Keep in mind the input for each node is either initial data from a flattened matrix, or an output from a previous node. The weight represents a value that is used as a measure of importance or the effect of a decision. Initially it is an arbitrary number that will be adjusted in training, and will be explained in Backpropagation [7]. The bais is an assumed constant value with the purpose of generalizing and offsetting data, and therefore will be added to the product of the weight and the input. The output layer consists of the final weighted summation of all contributing nodes and will represent the probability of the image being of a certain class.

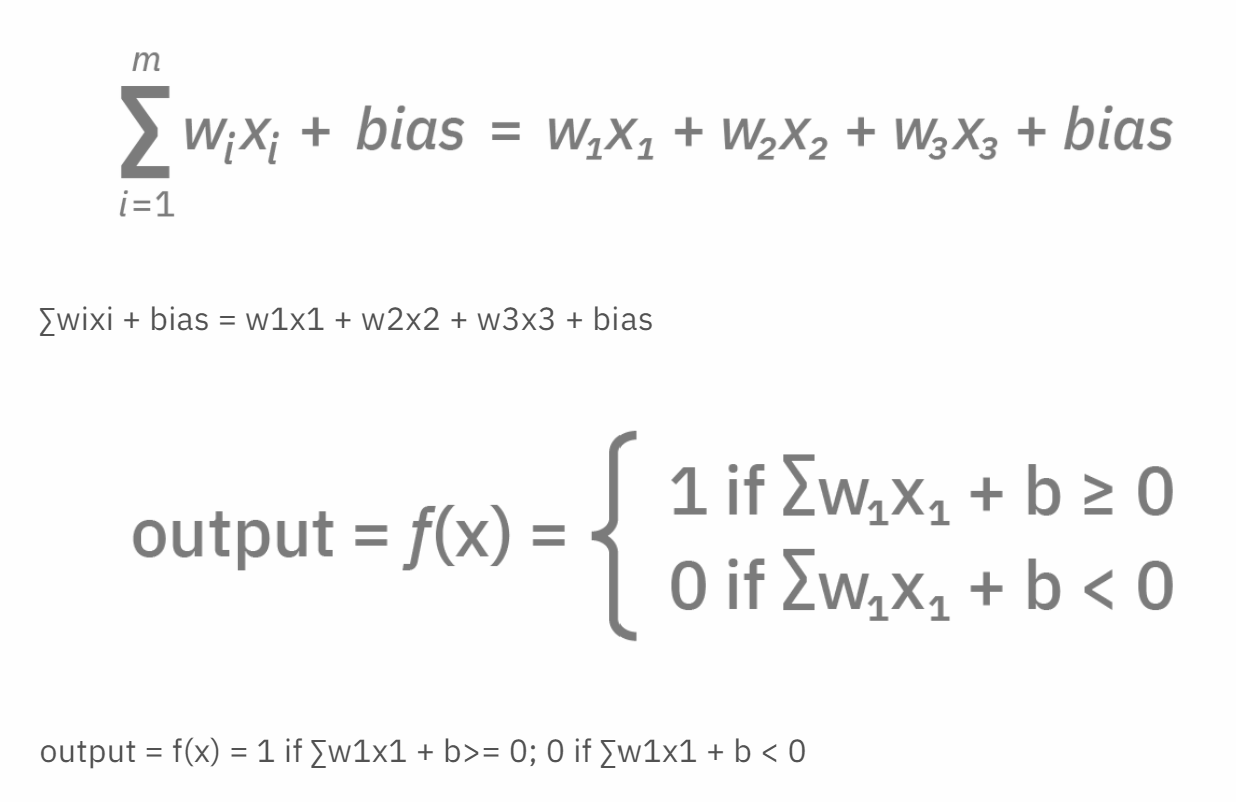


Figure 3 [7]

Feed-forward propagation is just the first half of neural networks. To create an effective and accurate classifier, training is heavily relied upon. Training includes calculating the error of the whole model and each node by using a cost function, also known as the mean squared error [4]. The mean squared error for each output and the total error will be key to backpropagation. Backpropagation is the common method for training a neural network. The end goal for this technique is to update each of the weights in the network [7]. Initially the weights are arbitrary values. By adjusting the weights to minimize the error in each neuron or node, there will be a reduction of the error for the network as a whole. The following steps come from the article “A Step by Step Backpropagation Example” by the software entrepreneur and former Air Force officer, Matthew Mazur.

Backpropagation begins at the output layer, specifically at the first output node shown in Figure 4. The goal is to update the weights in the network since they are initially arbitrary values. Therefore one needs to determine how much change the weight affects the total error of the network. This translates to the partial derivative of with respect to. In this example the first weight adjustment will be , and by applying the chain rule a main equation can be known as follows:

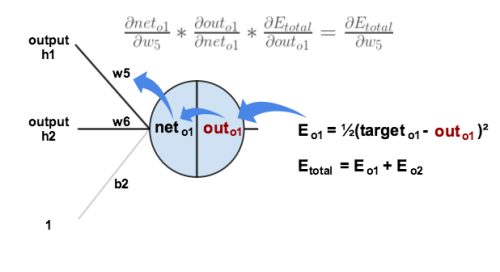


Figure 4 [10]

The next step is to break down each piece in the main equation. Starting with the leftmost piece, which is the partial derivative of with respect to . This is written as , where is the average of the output scores and is the output of focus.

To further simplify, the quantity of and becomes zero when the partial derivative is taken. The reasoning is that the two output nodes, and do not explicitly affect each other. Therefore the quantity including is reduced to a constant [10].

Next, deconstruct to find how much the output changes with the respect to its total net input. Remember that the output of a node is from the activated state, therefore the partial derivative of the activation function. The sigmoid function is used for this example [10].

Then for the rightmost piece of the equation; which represents the how much the total net input, or weighted sum, of the output node changes with respect to the weight[10]. Again since the second output node does not directly affect the result of the partial derivative transforms those variables to zero.

Putting together each simplified pieces’ equivalent into of the original outer layer formula for the first output node, can be alternatively extended into the following equation:

Finally to decrease error and calculate the new weight value, subtract the current weight by In this equation there is a variable n that represents the model learning rate, and it is optional [10]. This weight adjustment process may be repeated on the rest of the weights connected to an output node [10].

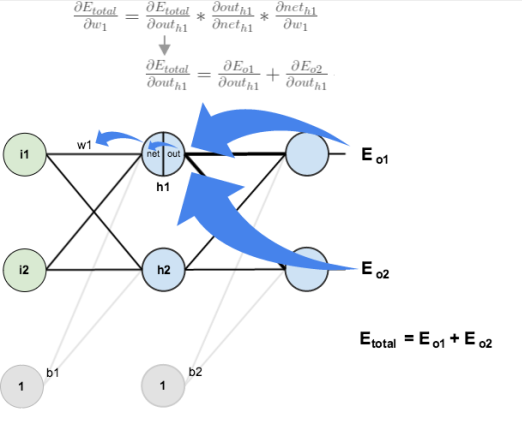


Figure 5 [10]

The process does not end here, the weights in the hidden layer must be adjusted too. The steps are similar but altered. This is due to the influence each hidden layer node has on its multiple connections, and those outputs and inputs must be considered. The main equation to update a weight that is connected to hidden layers follow this format [10]:

Start by solving each piece of the main equation, as seen previously. Begin to solve for. It becomes known that affects both and , thus the hidden node partial derivative equation will need to consider both pieces [10].

The original equation can be broken down further into another pair of equations [10]. Solve for both parts and .

However these subparts need to be broken down even further too. Solve for their left subpart and .

\*

\*

Then solve for the right subparts of the above equations. Its is known thatTherefore the partial derivative of the net values of outer layer one can be written as such [10];. It is also known that .Therefore . The first piece will look like this completely extended [10]:

Find the second piece of the original hidden layer equation, specifically . Remember that represents the output value, from the activated state, that is calculated using the sigmoid function. Therefore the partial derivative with respect to will result in the following equation [10].

Lastly solve for final part of the original hidden layer equation .This is the partial derivative of the total net input, the weighted sum from the preactivated state, for the hidden layer node with respect to the first weight value [10]. This will be one similar to what was done for the output node earlier.

Now that each pieces of the initial hidden layer equation has been broken down, substituting the extended equivalent equations for each piece of.

Similar to the output layer, update the weights [10]. Again in this equation there is a variable n that represents the model learning rate that it is often optional [10]. This weight adjustment process may be repeated on the rest of the weights connected in the hidden layers [10].

1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs), a specialized subset of neural networks, are very common in the classification of aspects or objects in an image, and can be trained to differentiate images from one another [11]. CNNs can be used to work with many different color spaces of images such as RGB, Grayscale, and HSV. However, this can be very computationally intensive and complex. It is therefore the job of the CNN to reduce images into a form that is easier to handle, but does not lose the key features that are critical to prediction unlike Feed-Forward Neural Nets. Neural nets alone take an image matrix and transform it into a flattened vector, and in doing so loses accuracy once sent onto a classifier [11]. The main difference between straight neural networks and convolutional neural networks is the added use of matrix manipulations or a window, and layers that process the data from the window. This window is typically called the Convolutional Layer or the Kernel. The kernel filter layer has the responsibility of traversing the input image, which is commonly sized to have an odd length and width, for example, the dimensions 5x5x1. The kernel itself is also typically odd in dimensions, such as 3x3x1 [11].

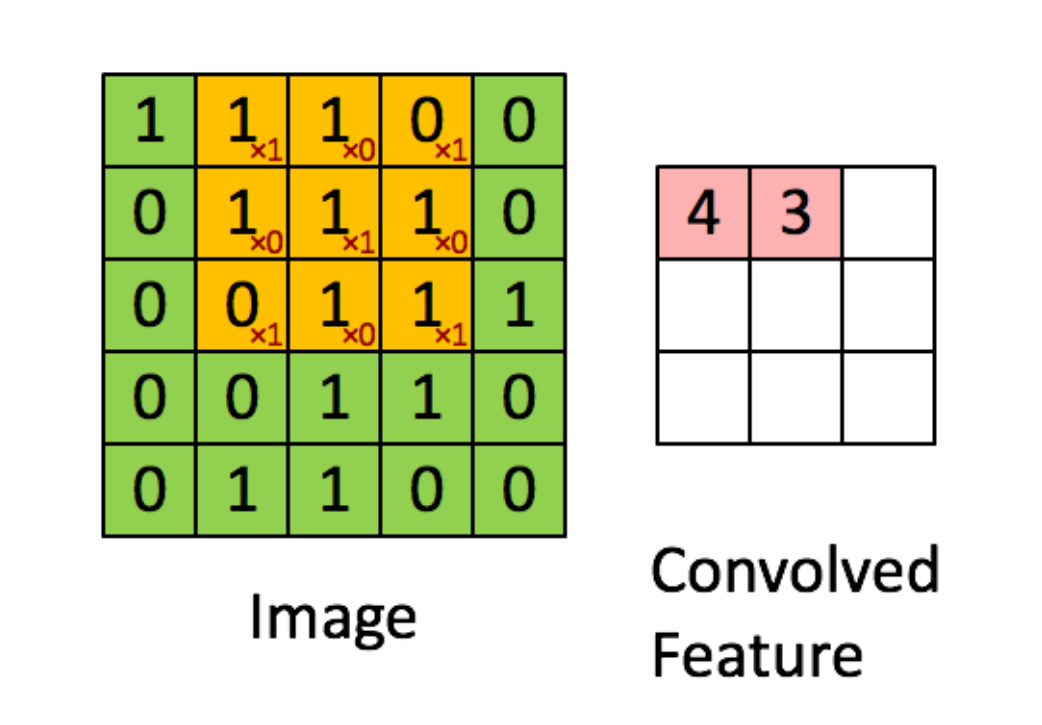


Figure 6 [11]

When the kernel is hovering over a portion of the image it will perform matrix multiplication operation between its filter, values that are weighted to represent a pattern, and the portion, then store the result in what is called a Convoluted Feature Output. The kernel will move one pixel to the right until it reaches the end of the image, then shifts down one and returns to the far left of the image. For the example in Figure 6, the kernel will make 9 shifts total and produce a 3x3x1 Convoluted Feature Output as seen in Figure 6 [11]. If the input is grayscale then only one feature matrix is needed. However, if the input has three channels such as RGB, then three feature matrices will be needed and then totaled together with a bias, typically with a value of one. This convolution type is called Valid Padding because the convolved matrix is reduced in size dimensionally in comparison to the input. The other padding type for the kernel layer is Same Padding. This technique allows for the kernel layer and the resulting convoluted feature matrix to be the same dimensions as the image [11]. This requires augmentation. For example, in Figure 6 if the original image is 5x5x1, it is adjusted to have a dimension 6x6x1 to create an empty space or padding for the kernel layer, which is still 3x3x1, to create a convoluted feature map that is the same size as the original input. Valid padding is the common approach as it does not add zeros to pooling data, but reduces the output [11].

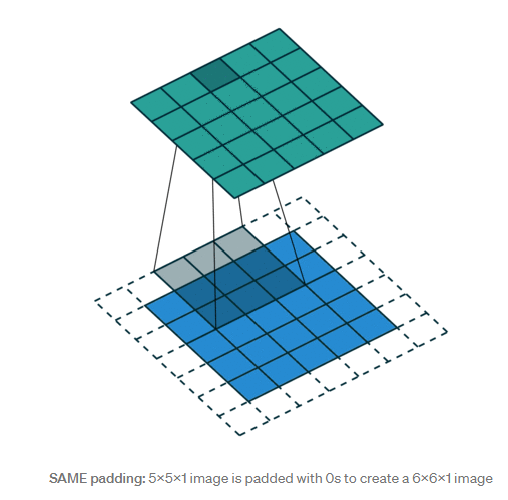


Figure 7 [11]

The next and last layer of the convolution series of layers in the Pooling Layer. More dimensional reduction happens here. It’s role is to decrease the computational power required to process the input data and to extract the dominant or key features. More specifically features that are invariant or unchanging to rotational and positional transformations [11]. Similar to the traversal in the convolutional layer, another matrix or pooling matrix will traverse over the convoluted feature matrix doing one of two pooling type methods. One type is Average Pooling, which returns the average of all data in the pooling window that is covering the feature matrix. The other type is Max Pooling which returns the maximum values within the covered area of the feature matrix. Both techniques perform dimensional reduction but only max pooling performs noise suppression, random variations in color or brightness [11]. The noisy activations or data is thrown out, while average pooling does not toss but incorporates the noise into the results. Therefore, max pooling tends to perform better and is used more often [11].

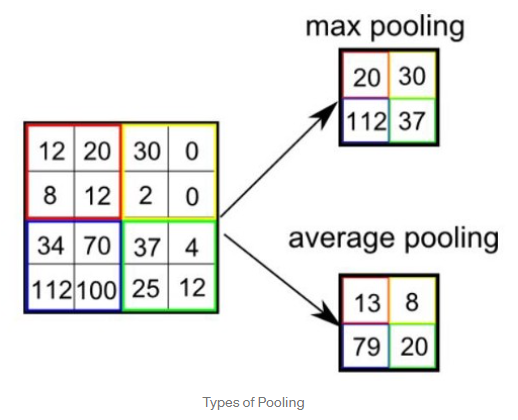


Figure 8 [11]

Altogether, building the convolutional layers and pooling layers, are fairly simple and typically paired together. Then the convolutional organization can be tacked onto the implementation of the neural net with the flattening of the last resulting feature matrix as the input. From there feed forward traversal of the network can be applied to the hidden layers within the neural network portion [11].

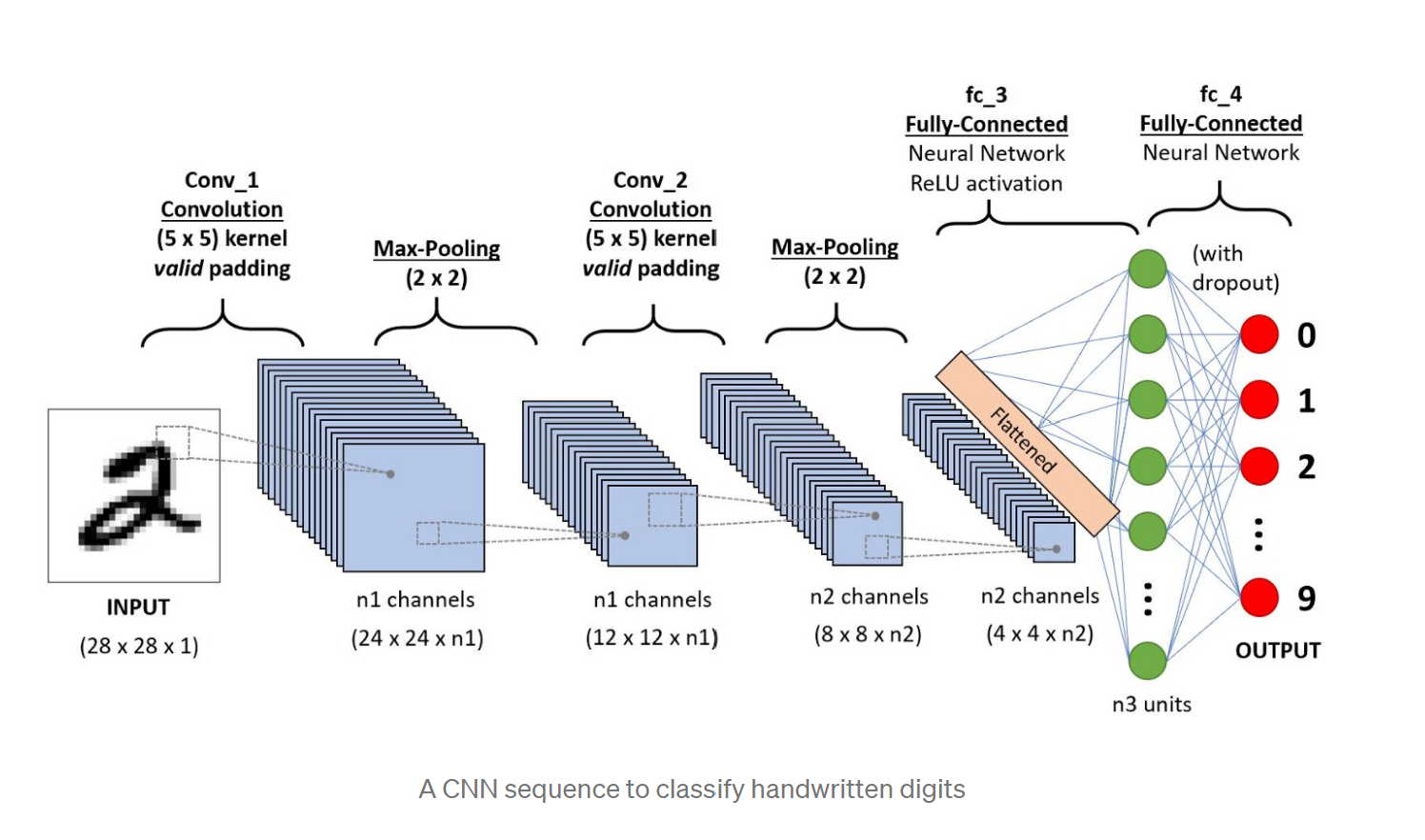


Figure 9 [11]

1. Primary and Secondary Objectives

*A. Primary Objective*

Access how scaling of images with single objects, affects the accuracy of a CNN classifier.

*B. Secondary Objective*

Explore the effects on accuracy of a CNN classifier with additional convolutional layers.

1. Solution Description

Models were built using the programming language Python within the free source-code editor Visual Studio Code. Free open-source libraries for machine learning, TensorFlow, Keras, and Scikit-Learn were used.

The dataset used is the CIFAR-10 image dataset. It contains 60,000 32x32 color images which are predefined into ten different classes: classes: The categories are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The dataset is also made up of 50,000 training images and 10,000 test images for simplified splitting of data.

The 32x32x3 dataset was reduced twice by averaging pixels under a window with a size of 2x2, to produce 16x16x3 images, and a window of 4x4 to produce 8x8x3 images

Next two CNN models were created. The first is referred to as the *Standard* in this paper for it is used to train and test on 32x32, 16x16, and 8x8 image datasets. It contains 2 convolutional layers, the first layer will have 32 filters, and the second layer has 64 filters. The kernels will be activated as filters by Keras’ Glorot uniform initializer, and are of size 3x3 on the first layer and size 2x2 on the second. Image padding will be ‘same’, and the kernel filters move in strides of one. The activation functions will be ReLu. The second CNN model is referred to as the *Research* in this paper and is used to train and test on 32x32 and 16x16 image datasets. The layers have 32, 32, and 64 kernel filters in that order, with ‘same’ padding and uses ReLu as the activation function. The connecting neural network for both models uses a max pooling kernel of 2x2, has a dropout layer with a rate of 0.5, and has a final dense layer with Softmax activation function.

1. Hypotheses & Goal Tree

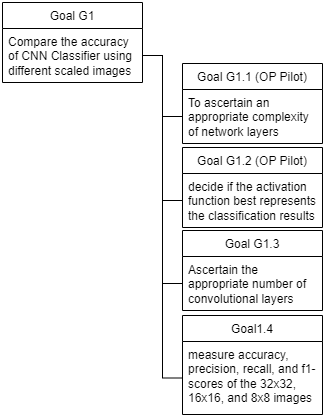


Figure 9

1. *Hypothesis 1*

Scaling an image by ½ has no effect on performance.

1. *Hypothesis 2*

Scaling an image by ¼ has no effect on performance

1. *Hypothesis 3*

There is no difference in performance between a CNN model with 2 layers and a CNN model with 3 layers.

1. Experiment Design

| Factor | Values |
| --- | --- |
| Images | Single object, classes mutually exclusive |
| Scaling | N, N/2, N/4 |
| CNN Filters | “Standard Model” of 2 Layers  “Research Model” of 3 Layers |
| samples | 60,000 images broken up into a training group of 50,000 and 1 test group of 10,000 images. |

Figure 10

There are five experiments that were conducted: images of size 32x32 - 2 layers, 16x16 - 2 layers, 8x8 - 2 layers, 32x32 - 3 layers, and 16x16 - 3 layers.

1. Results

Figures 11 and 12 show the averages and standard deviations of the five experiments for precision, recall, and f1-score. Precision is the number of correctly identified positively classified instances divided by the total number of positive predictions made. In other words, it shows the number of correct positive predictions. The recall is the measure of how many positive instances were correctly predicted over all of the positive instances. Finally, f1-score is described as the combination of precision and recall to produce a harmonic mean, or an average of two ratios, with the intent to provide a single metric that is balanced [12].

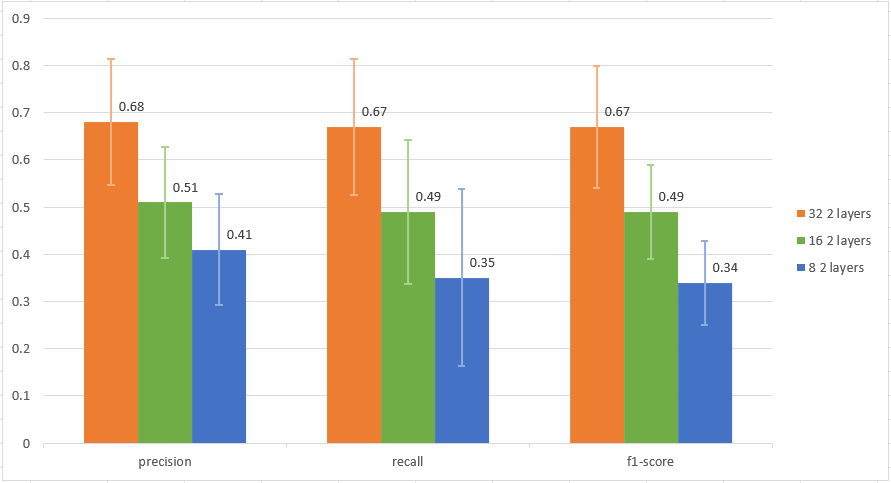


Figure 11

Looking at the comparisons of the models with two layers there is an overall downward trend in the averages of precision, recall, and f1-score. Higher measurements in these three areas suggest an effective model, and the decrease may suggest that scaling the size of an image may in fact worsen the performance of the CNN model [12]. Especially since the standard deviations between 32x32-2 and 8x8-2 have a very small or limited overlap in values. However, 16x16-2 has a standard deviation that overlaps with 32x32-2 and 8x8-2 which may suggest that scaling an image by ½ does not have a strong effect on model performance.

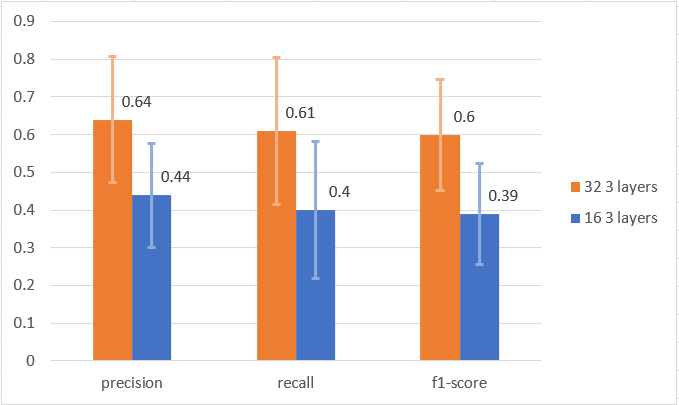


Figure 12

In figure 12 a decrease in the averages are present, however there is a decent amount of overlap between the standard deviations of 32x32-3 and 16x16-3. This may suggest that there is no difference in performance between scaling images by ½ and increasing the number of layers as the averages are close to those with two layers in figure 11. However to further analyze the data a t-Test is used to visualize and measure if the experiment findings are significant, close in relation, and to quantify whether the results are likely due to chance or to some factor of interest.

| t-test | | | | |
| --- | --- | --- | --- | --- |
|  | | precision | recall | f1-score |
| 8-2 vs. 16-2 | | 0.07689 | 0.07815 | 0.00348 |
| 8-2 vs 32-2 | | 0.00018 | 0.00052 | 0.00001 |
| 16-2 vs. 32-2 | | 0.00889 | 0.01593 | 0.00256 |
| 16-3 vs. 32-3 | | 0.00857 | 0.02239 | 0.00322 |
| 32-2 vs. 32-3 | | 0.45358 | 0.5284263511 | 0.293665295 |
| 16-2 vs. 16-3 | | 0.2202120616 | 0.2268365364 | 0.08452733341 |

Figure 13

Figure 13 shows the results of two-sample t-tests, with unequal variance and two-tailed distributions. The left hand labels denote the image dimensions as a single integer followed by an integer that represents the number of layers in the CNN model, for example a 2 layer CNN that trains and tests on 8x8 images will be labeled as ‘8-2’. The highlighted values that are dark green show a level of significance (1 - confidence level of 0.95) as they are less than 0.05.

*A. Hypothesis 1*

Hypothesis 1 can be examined by the following experiment comparisons: 8-2 vs. 16-2, 16-2 vs. 32-2, , and 16-3 vs. 32-3. The first experiment has a significant score in f1-score, but the precision and recall t-scores are not so significant being a few points greater than 0.05. The remaining two experiments in this grouping have extremely high significant levels, puting the confidence level at approximately 98%. This suggests that there is a negative effect on performance.

*B. Hypothesis 2*

Hypothesis 2 can be examined by the experiment comparison: 8-2 vs. 32-2. The precision, recall, and f1-scores all have small values well beneath the significance level, meaning that the confidence level is approximately 99%. This suggests that reducing an image by ¼ could affect the model, and in a way that is negative.

*C Hypothesis 3*

Hypothesis 3 can be examined by the following experiment comparisons: 32-2 vs. 32-3, and 16-2 vs. 16-3. Both of the experiments resulted in t-scores for precision, recall, and f1-scores that were all greatly above the significance threshold of 0.05. Therefore the confidence level is very low, suggesting that the addition of one convolutional layer into the models had little to no effect on performance.

1. Conclusions

Hypotheses 1 and 2 can be rejected whereas hypothesis 3 fails to be rejected. In other words, it is supported that reduced scaling has an effect on performance, and increasing the number of layers in a CNN model has no effect on the performance is inconclusive and requires further testing.

The limitations of this project include the initial image dataset having samples that are extremely small with the size of 32x32. The dimensions restrict the number of convolutional layers that may be used and the quality of the images appear questionable. When using Keras, TensorFlow, and Scikit-Learn there is the issue of the libraries offering code that are a ‘black-box’. The library methods and objects are built often with default functions and values for function parameters, which hides a lot of intializiers from the programmer.

REFERENCES

[[1] “Basics of Machine Learning Image Classification Techniques,” *OpenGenus IQ: Learn Computer Science*, Aug. 15, 2019. https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/ (accessed Mar. 30, 2021).](https://www.zotero.org/google-docs/?1k2X6e)

[2] D. Nelson, “How Does Image Classification Work?,” Unite.AI, Sep. 05, 2020. https://www.unite.ai/how-does-image-classification-work/ (accessed Apr. 05, 2021).

[3] P. Madhu, R. Kosti, L. Mührenberg, P. Bell, A. Maier, and V. Christlein, “Recognizing Characters in Art History Using Deep Learning,” in Proceedings of the 1st Workshop on Structuring and Understanding of Multimedia heritAge Contents - SUMAC ’19, Nice, France, 2019, pp. 15–22, doi: 10.1145/3347317.3357242.

[4] T. Liu, S. Fang, Y. Zhao, P. Wang, and J. Zhang, “Implementation of Training Convolutional Neural Networks,” p. 10.

[5] “Digital Image Processing Basics,” GeeksforGeeks, Jan. 26, 2018. https://www.geeksforgeeks.org/digital-image-processing-basics/ (accessed May 02, 2021).

[6] S. K, “Introduction to Digital Images,” Medium, Apr. 30, 2021. https://towardsdatascience.com/introduction-to-images-c9c7abe6bfd2 (accessed May 02, 2021).

[[7] “What are Neural Networks?,” Jan. 06, 2021. https://www.ibm.com/cloud/learn/neural-networks (accessed Mar. 24, 2021)](https://www.zotero.org/google-docs/?1k2X6e)**.**

[[8] Vikashraj luhaniwal, “Forward propagation in neural networks — Simplified math and code version,” Medium, Oct. 24, 2020. https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250 (accessed May 09, 2021).](https://www.zotero.org/google-docs/?1k2X6e)

[9] V. Jain, “Everything you need to know about ‘Activation Functions’ in Deep learning models,” Medium, Dec. 30, 2019. https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253 (accessed May 13, 2021).

[10] Mazur, “A Step by Step Backpropagation Example,” Matt Mazur, Mar. 17, 2015. https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (accessed Apr. 21, 2021)

[11] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science,” Dec. 15, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed Mar. 27, 2021).

[12] T. Kanstrén, “A Look at Precision, Recall, and F1-Score,” Medium, May 19, 2021. https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec (accessed Dec. 10, 2021).

[1] C. Park, Image Classification with CIFAR-10 dataset. 2021. Accessed: Dec. 11, 2021. [Online]. Available: https://github.com/deep-diver/CIFAR10-img-classification-tensorflow

[14] “Convolutional Neural Network (CNN) | TensorFlow Core,” TensorFlow. https://www.tensorflow.org/tutorials/images/cnn (accessed Dec. 11, 2021).