

Image Classification with Convolutional Neural Networks

Kenyon LeBlanc and Dr. C. Dudley Girard

Shippensburg University

kl2509@engr.ship.edu and cdgira@cs.ship.edu

ABSTRACT

This paper delves into the complexities of automating image classification using convolutional neural networks (CNNs). Image classification is a significant aspect of computer vision, but traditional models like SVMs and Random Forests can struggle with accurate feature extraction due to factors like camera angles and background clutter. CNNs offer a promising solution to this challenge. The paper begins with an introduction to image classification and the architecture of convolutional neural networks. Preprocessing techniques such as normalization and color conversion are examined, emphasizing their role in enhancing model performance. CNN layers are highlighted and explained with equations and visual aids to clarify concepts such as filters, weights, biases, and activation functions. Additionally, the importance of forward and backward propagation in training the neural network is highlighted along with techniques such as dropout regularization. The overall theme of this paper is to give an understanding to the processes and methodologies of convolutional neural networks.

KEY WORDS

Convolutional Neural Network * Classification * Kernel Backpropagation

1. Introduction

Image classification consists of categorizing images with labels or into groups according to their category. It is not an easy process to automate. There are many factors to consider such as camera angles, background clutter, and where the subject is placed in the image. These factors can impact some machine learning models such as SVMs and Random Forests which rely on accurate features being extracted. A solution to tackle this problem is utilizing a convolution neural network (CNN). CNNs are useful in classifying images due to their method in extracting image features using kernels (filters) on pixels. It then ranks these filters in a hierarchical manner, placing a higher importance on some filters over others due to some common features being found in an image.

Before a set of images can be plugged into a CNN, they must first be preprocessed. This process consists of normalization, interpolation, and sometimes color conversion. After the preprocessing, images will be inputted into a CNN. This is when the process called

forward propagation is initiated. The first set of layers that will be used are the convolution and pooling layer

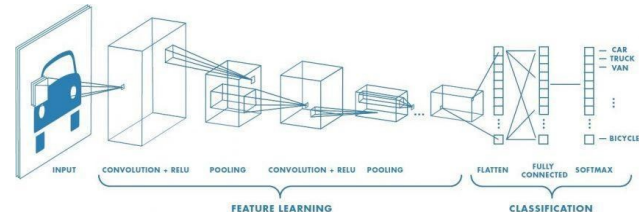


Figure 1: The architecture of a CNN [12]

(see Figure 1). These are feature extraction layers. The convolution layer applies either static filters or learnable filters to an image. The pooling layer takes a group of pixel values and picks the most important one to keep. These two layers tend to down sample an image. Some models use padding to counteract this, however other models intentionally do this. After the feature extraction layers and before the classification layer, resides the flatten layer. This converts the multi-dimensional maps into a one-dimensional vector. This conversion makes the data suitable for the fully connected layer. Each individual data point in the vector is connected to a neuron in the fully connected layer. When data is manipulated through this layer, it is then off to the final layer, the output layer. This is the layer that figures out the probability per category. Each layer has weights and biases which are then altered depending on the results of forward propagation. Backwards propagation essentially changes the weights and biases to minimize the loss function. This is to enhance the accuracy of the CNN model. All these layers and processes in a CNN are used to achieve an accurate image classifier.

2. Background

2.1 Preprocessing

Before a set of images can be used in a convolutional neural network, they must be preprocessed. This process formats the input data with a standard all the data must follow. This entails scaling pixel values to a common range, formatting the dimensions of data to a common size, and potentially modifying images in other ways that help improve performance from the model. This is important because throughout the CNN, feature maps will be gathered and continually shrink. The data needs to be standardized so these feature maps stay consistent.

2.2 Normalization

Normalization is a preprocessing technique in image processing that involves scaling input data, for example pixels, into a specific range. This scaling ensures that the data falls within a certain numerical range, which can be $[-1, 1]$ or $[0, 1]$, depending on the requirements of the model [1]. Potential issues coming from varying scales can be mitigated by normalizing the pixel values. Thus, aiding in achieving the models optimal state during training. Image interpolation is another crucial preprocess

2.3 Image Interpolation

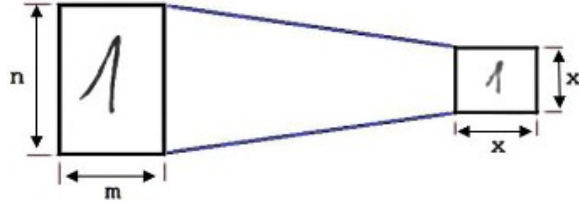


Figure 2: Example of image interpolation. [1]

step in image manipulation. The process particularly deals with resizing images while preserving essential information (see figure 2). Given the abundance of pixels in an image, downsizing them without losing vital visual features is crucial. Image interpolation techniques ensure that the resized image maintains its integrity by accurately estimating pixel values between existing data points [1]. This preservation of information is vital for image classification tasks, where precise details are crucial for accurate classification.

2.4 Color Conversion

Converting colors is a preprocessing step that can be beneficial in certain scenarios. Color information may not significantly contribute to some tasks. While RGB representation is standard, converting complex color values to a grayscale range between $[0, 1]$ (see figure 3) or a binary format (0 or 1) can enhance model performance [12]. Computational resources can be

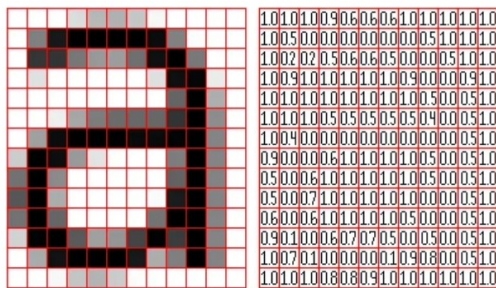


Figure 3: Pixel values on the left with the corresponding grayscale values on the left. [12]

conserved by simplifying color representations. Potential noise from irrelevant color channels can also be reduced, leading to improved model efficiency and accuracy.

2.5 Transforming Data

Transforming data through techniques such as flipping, rotating, and cropping can further enhance preprocessing efforts. For instance, random cropping of rescaled images, combined with random horizontal flipping, and random RGB color and brightness shifts, are commonly employed strategies [9]. These transformations not only introduce diversity into the training data but also enhance the robustness of the model by exposing it to variations commonly encountered in real-world scenarios [9].

2.6 Convolution Layer

The convolution layer serves as the first layer where input data is manipulated in a CNN. It operates by utilizing several filters (kernels) that scan over the image. These filters are typically 2×2 or 3×3 matrices. The stride value determines the number of values it moves over at a time. As it scans over the input data (image), each filter matrix multiplies with the corresponding input data matrix (see figure 4). The individual values in the filter

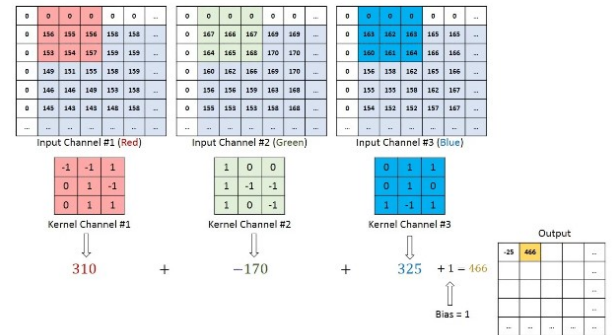


Figure 4: An example of how the convolutional layer computes output with a depth of 3. [7]

represent the weights of the layer. Additionally, a bias value, one value per kernel, is added to each resulting point before summing all outputs. This single bias value helps mitigate underfitting, a problem where the model cannot find a relationship between inputted data and its output [18]. After computing the output matrix, the size will shrink. The matrix size can be calculated with equation 1 below.

$$(1) \quad N = W - F + 2PS + 1 \quad [10]$$

Equation for determining output size of matrix N. Variable W represents the input size of the feature map, F is the convolutional kernel size, P is the padding size, and S is the step/stride value.

2.7 Filter

Filters, also known as kernels, are typically 2×2 or 3×3 matrices used in convolution layers. A convolution layer may contain numerous filters, each trained to detect different features within the input data. These filters play an important role in feature extraction and detection, such





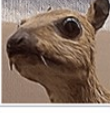
Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Figure 5: Examples of static filters. [7]

as identifying edges or textures in images. Filters can also have static values and do not possess weights and biases like traditional neural network components (see figure 5). They are versatile and find applications in various image processing tasks, including edge detection [7].

2.8 Weights and Bias

Weights represent the strength of connections between neurons in a neural network. They are continually modified through computed loss functions during backward propagation to enhance the accuracy of the

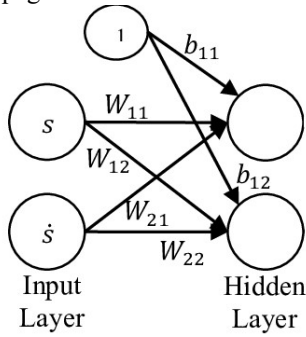


Figure 6: Visual representation of weights and bias values in a trained network. [5]

model. Bias, on the other hand, is a randomly initialized value that helps a model remain flexible. If an input value is zero and the model was without a bias, the result of the value scaled by the weight would always be zero. Bias adds an independent factor that prevents the model from solely relying on input data [10]. See figure 6 for a visual representation of weights and biases. Regarding the convolutional layer, each value in the kernel represents a weight and each filter has a single bias value (see figure 4).

2.9 Activation Function

An activation function is often used on resulting data before passing it to the next layer. It is crucial for handling non-linear resulting data and countering the vanishing gradient problem. The Rectified Linear Unit

$$(2) \quad f(z) = \max(0, z) \quad [15]$$

The ReLU function either outputs a zero if the value is negative or whatever the raw input is.

$$(3) \quad f(z) = \frac{1}{1+e^{-z}} \quad [10]$$

The Sigmoid Function transforms raw inputs into a value between zero and one.

$$(4) \quad f(z)_i = \frac{e^{z_i}}{\sum_k e^{z_k}} \quad [15]$$

Softmax function takes raw inputs and transforms them into probabilities totalling to one.

(ReLU), equation 2, is a commonly used activation function for all layers. The sigmoid function, equation 3, is mainly for binary classification tasks. Additionally, the Softmax function, equation 4, is used in the output layer for classification purposes. Softmax is used on the results in the output layer, confirming they add up to 1 [15]. These activation functions ensure mathematical stability and prevent learned values from getting stuck near zero or blowing up toward infinity [1][6][10][12][14][18].

2.10 Pooling Layer

The pooling layer serves as a pivotal component in convolutional neural networks (CNNs), significantly reducing the dimensions of the data while retaining essential information. As the layer pans over the input data, typically with a 2x2 matrix window, it selects either the maximum or average value. The impact of the pooling layer extends across multiple channels, with the depth of the data influencing the number of maps affected. For instance, in RGB data there are three values representing three different channels. The pooling layer must consider each channel's information while downsizing the spatial dimensions. Its primary objective lies with preserving the most crucial features of the input data, not just the dimensionality reduction. As explained by [18], the pooling layer effectively takes large images and compresses them while safeguarding essential information. Thus, ensuring the maximum value or average fit within each window is retained.

Max pooling selectively preserves the most prominent features during downsizing. As the window traverses the input data, only the maximum value within each window is retained. See figure 7 for visual representation of the

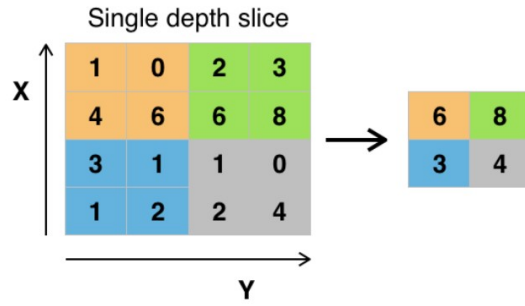


Figure 7: An example of 4 filters on the left computing max pooling, with the output on the right. [2]

process. This strategy, documented in [2] and [10], ensures that crucial details are preserved, contributing to the network's ability to recognize and extract essential patterns from the data. In contrast to max pooling, average pooling offers a different approach. As the window moves across the input, average pooling calculates the mean value within each window. This leads to a more even distribution of information. This technique, supported by [2] and [10], ensures that the resulting feature map reflects a more balanced representation of the input data.

2.11 Flattening Layer

The flatten layer serves as a pivotal transition point in CNN architecture. Unlike the computational layers preceding it, the flatten layer does not perform any computations. It reshapes the data into a single flat vector, preserving the spatial information. It is done by taking all the extracted feature elements from the input tensor into a single dimensional array. This flattened vector serves as the input for subsequent fully connected layers, facilitating the propagation of information across the network [13].

2.12 Fully Connected Layer

In neural networks, the fully connected layer incorporates high-level features extracted from previous layers. Each neuron in this layer establishes connections with every neuron from the preceding layer. The first layer consists of the flatten layer, which organizes the data into a flat vector [13]. Through these connections, the fully connected layer combines features to generate an output. This process involves the application of weights, biases, and an activation function, ultimately yielding probability scores corresponding to distinct image classes [19]. During the forward pass, each neuron computes the weighted sums of their respective connections to previous neurons. This net result is then inputted into an activation function to introduce non-linearity. This process is to transform the raw inputs from the flatten layer and manipulate them in such a way to achieve probability scores for the neurons in the output layer.

2.13 Output Layer

In image classification tasks, the output layer determines the class probabilities for the given input. With multiple classes to consider, the number of neurons in this layer corresponds to the number of classes, with each neuron representing a distinct class [23]. The primary function of this layer is to assign scores to each class, indicating the likelihood of the input belonging to a particular category. As the data typically undergoes normalization within the preceding layers, often via activation functions like sigmoid (see formula 3 and section 2.10), the output layer may not necessarily require an additional activation function for binary classification tasks. However, for the purpose of classifying multiple animal species, the softmax activation function (see formula 4 and section 2.10) will be utilized to attain a 100% percentage split between multiple output neurons.

2.14 Forward Propagation

Forward propagation involves passing input data through the layers to produce an output. When an image is fed into the network, each pixel's value is represented by an activation in the first layer. These activations are weighted, summed, and activated with an activation function in subsequent layers. Each neuron in these layers

$$(5) \quad a_0^{(L)} = f(w_{0,0} \cdot a_0^{(0)} + w_{0,1} \cdot a_1^{(0)} + \dots + w_{0,n} \cdot a_n^{(0)} + b_0^{(L)}) \quad [17]$$

A function for getting the activated neuron value where $a_0^{(L)}$ represents the output for the first neuron in layer L, f represents the activation function, $w_{0,0}$ represents the weight of the connection between neurons, $a_0^{(0)}$ represents the activated neuron value in previous layer, and $b_0^{(L)}$ represents the bias value. See figure 9.

$$(6) \quad a_j^{(L)} = f\left(\sum_i^n w_{ji} \cdot a_i^{(L-1)} + b_j^{(L)}\right) \quad [17]$$

Condensed function of equation 5.

$$(7) \quad a^{(L)} = f(w^{(L)} \cdot a^{(L-1)} + b^{(L)}) \quad [17]$$

A function for getting the layer of activated neuron values where $a^{(L)}$ represents the activation layer, f represents the activation function, W represents the weight matrix for connections between layers, $a^{(L-1)}$ represents the previous layers activation value matrix, and b is the layers bias. See figure 8.

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Figure 8: Visual for equation 7. The first matrix on the left represents the weights between neurons, to the immediate right are the activation values in a vector, finally on the far right are the bias values for the activation neurons in a vector. [17]

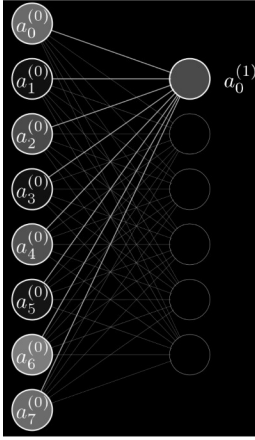


Figure 9: Visual for equation 5. This figure shows the activation neurons in layer (0) and weighted connections used to compute $a_0^{(1)}$. Note the activation function and bias value is not displayed but needed for the equation. [17]

represents different features of the input. The activations are transformed using functions like sigmoid in the fully connected layer or SoftMax in the output layer. In this case, each neuron corresponds to a possible animal. The network essentially acts as a complex mathematical function with thousands of parameters that are adjusted during training to optimize performance in image classification [17].

2.15 Backwards Propagation

Critical to the training process of neural networks, backwards propagation enables iterative updates to network weights with the goal of minimizing loss. This methodology involves computing the gradient of the loss function concerning each weight, employing the chain rule to propagate errors backward through the network layers [2]. By iteratively traversing backward from the

$$(8) \quad w_k^{(L)} = w_k^{(L)} - \alpha \frac{\partial C_0}{\partial w_k^{(L)}} \quad [3]$$

The equation represents the updated weights of layer L during training, where α is the learning rate and $C_0 w_k^{(L)}$ is the partial derivative of the cost function with respect to the partial derivative of weights.

$$(9) \quad b_k^{(L)} = b_k^{(L)} - \alpha \frac{\partial C_0}{\partial b_k^{(L)}} \quad [3]$$

The equation represents the updated biases of layer L during training, where α is the learning rate and $C_0 b_k^{(L)}$ is the partial derivative of the cost function with respect to the partial derivative of biases.

output layer, redundant calculations of intermediate terms are avoided, streamlining the optimization process and enhancing training efficiency [2].

3. Primary Objective

The primary objective of this research is to assess how activation functions and learning rate impact the performance of a convolutional neural network (CNN) in accurately identifying specific animal species depicted in images. Activation functions introduce non-linearity into the CNN, enabling it to learn complex patterns and representations from raw pixel data. Learning rate

controls the rate at which the model updates its parameters during training. By evaluating the different combinations of activation functions and learning rates, this research aims to determine the optimal combination that leads to the highest accuracy in animal species identification tasks.

4. Solution Description

Python served as the programming language for this research. Packages and libraries were essential to achieve the construction of the CNN. NumPy, recognized for its efficiency, assisted with fast and precise matrix multiplication and data manipulation. Matplotlib was used in graphing experimental data and displaying images. Image interpolation and image transformation was handled by Albumentation during preprocessing. CSV, known for ease of interacting with csv files, was utilized for reading and writing data to files. Pickle was implemented to save and load models directly from memory. Additionally, predefined filters were carefully selected to extract appropriate features in the convolutional layer. Specifically, the edge detecting filters Sobel X, Sobel Y, Scharr X, Scharr Y, and Laplacian were chosen and implemented into the network to extract features. While a diverse array of datasets is available on platforms like Kaggle, the CIFAR-10 dataset stood out for containing a large set of standardized training and testing images per class across various animal species. Specifically, the 6 classes bird, cat, deer, dog, frog, and horse will be used. Each class will consist of 5,000 32x32 rgb training images and 1,000 32x32 rgb testing images. Classes do not distinguish between types. For example, the dog's class will not distinguish between labrador or chihuahua. It is also worth noting that TensorFlow was used to import the Cifar10 dataset directly into the program.

5. Hypotheses

Hypothesis 1:

- *Null Hypothesis:* Activation function will have no effect on performance.
- *Alternative 1:* ReLU activation function will outperform TanH and Sigmoid.
- *Alternative 2:* TanH activation function will perform the worst.

Hypothesis 2:

- *Null Hypothesis:* Learning rate will have no effect on performance.
- *Alternative 1:* 1e-05 learning rate will outperform 1e-03 and 1e-04.
- *Alternative 2:* 1e-03 learning rate will perform the worst.

6. Goal Tree

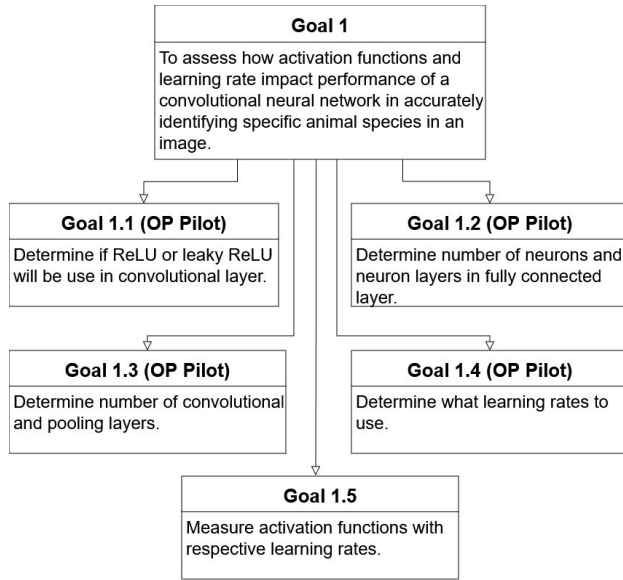


Figure 10: Goal tree

7. Experiment Design

Factors used in the experiment consist of activation functions and learning rates, seen in figure 11. The chosen activation functions are ReLu, Tanh, and Sigmoid while the learning rates are 1e-03, 1e-04, and 1e-05. Each unique combination of activation function and learning rate will be tested 5 times and average these 5 tests. It is important to note that each of these 5 tests under the same unique combination of factors will consist of a newly

Factors	Values
Activation Function	{ReLU, Tanh, Sigmoid}
Learning Rate	{1e-03, 1e-04, 1e-05}

Figure 11: Table of factors with respective values.

trained model under the same conditions. Each test takes the average of 10 accuracies attained through 10 batches of 600 test images. The 10 batches will give insight into how the models are performing between different ratios of images. More importantly this gives more data points to work with. With a training set of 30,000 32x32 rgb images, 1 convolutional and pooling layer, 1 neuron layer in the fully connected layer, and around 1 hour per training, it took approximately 45 hours to train all 45 models. With a testing set of 6,000 32x32 rgb images separated into 10 batches of 600 images and classified using one of the 45 pre-trained models, it took approximately 20 hours to gather all data.

8. Results

To start, figures 12 and 13 show the accuracy of models with a 95% confidence interval with added lines representing a statistical difference in accuracy gained through a paired t-test.. All following t-tests conducted use a critical t-value of +/- 2.776. The results were first analyzed to check whether or not there was a statistical difference when comparing to random chance. Random chance is 0.1667. Performing an one sample t-test using

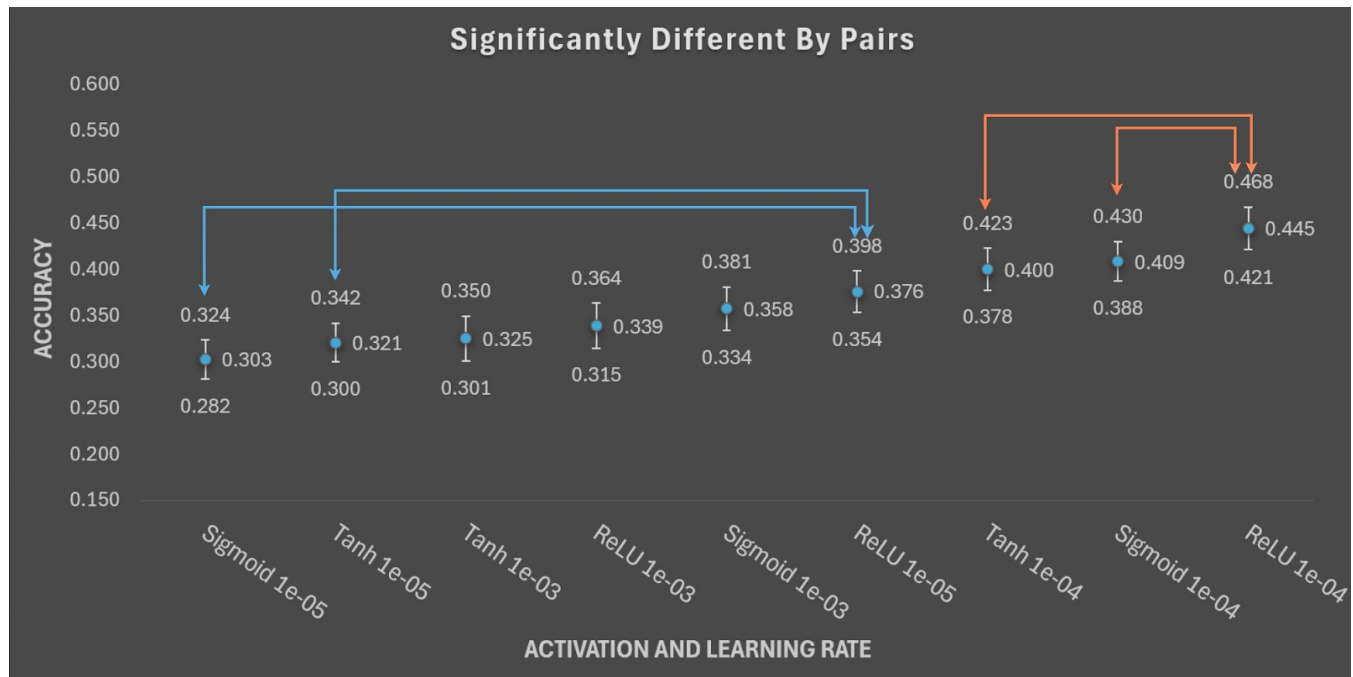


Figure 12: CNN model accuracy results with 95% confidence intervals with lines representing a statistical difference between accuracy.

the different accuracies to random chance, it was determined all results were statistically different. This means all models were likely different from random chance. Figure 12 shows the results from the t-tests that tested models with different activation functions with the same learning rate. There are no significantly different pairs between sigmoid and tanh, meaning neither is likely better. ReLU was found to be likely better with 1e-04 and 1e-05 learning rate, concluding ReLU performed the best among the activation functions. Figure 13 shows the results from t-tests that tested models with different learning rates with the same activation function. ReLU with the learning rate 1e-04 was statistically different from all other ReLU models with the learning rates 1e-03 and 1e-05. ReLU paired with 1e-05 was also significantly different from ReLU with 1e-03. This means that among the ReLU models, 1e-04 performed better than 1e-05 and 1e-05 performed better than 1e-03. It should be noted that the t-test result produced from ReLU 1e-05 and 1e-03 produced a result on the border of the critical t-value. Tanh at 1e-04 was significantly different from 1e-03 and 1e-05, meaning 1e-04 performed better than all other learning rates. Sigmoid at 1e-04 was likely better than all other learning rates, as well as 1e-03 likely better than 1e-05. The conclusion was made that there was not enough difference between the learning rates 1e-05 and 1e-03. The difference between ReLU at 1e-05 and 1e-03 was statistically significant but it represented the weakest possible evidence due to being on the critical threshold. Sigmoid at 1e-03 and 1e-05 was statistically significant, so further testing would need to be conducted to conclude anything further. However, the learning rate 1e-04 did perform the best.

9. Conclusion

Convolutional neural networks are used to achieve automated image classification. Preprocessing consists of methods such as normalization, image interpolation, and color conversion. This process is important because it reduces complexity the model must take into account by standardizing data, keeping consistency, and helping promote efficiency in the network. The convolution layer is where feature and pattern extraction is handled with the use of filters and weights. Afterwards the pooling layer downsizes data while keeping the important features. The flatten layer takes the data from the tensors and converts it into a single flat vector. This vector holds the neurons that are connected to the neurons in the fully connected layer, where the extracted features are processed. Finally, the output layer is where the probability scores are produced. These probabilities are what lets a convolutional neural network come up with an answer for what class an image should belong to based on features and patterns found in an image. Pivoting to the results, ReLU performed the best among the activation functions. It was also found that there was no significant difference between sigmoid and tanh. Among the learning rates, 1e-04 performed the best. Further testing would need to be conducted to determine whether 1e-03 and 1e-05 had enough of a statistical difference to conclude if one was better than the other. By utilizing machine learning techniques, the process of automating image classification can be achieved through the use of a convolutional neural network.

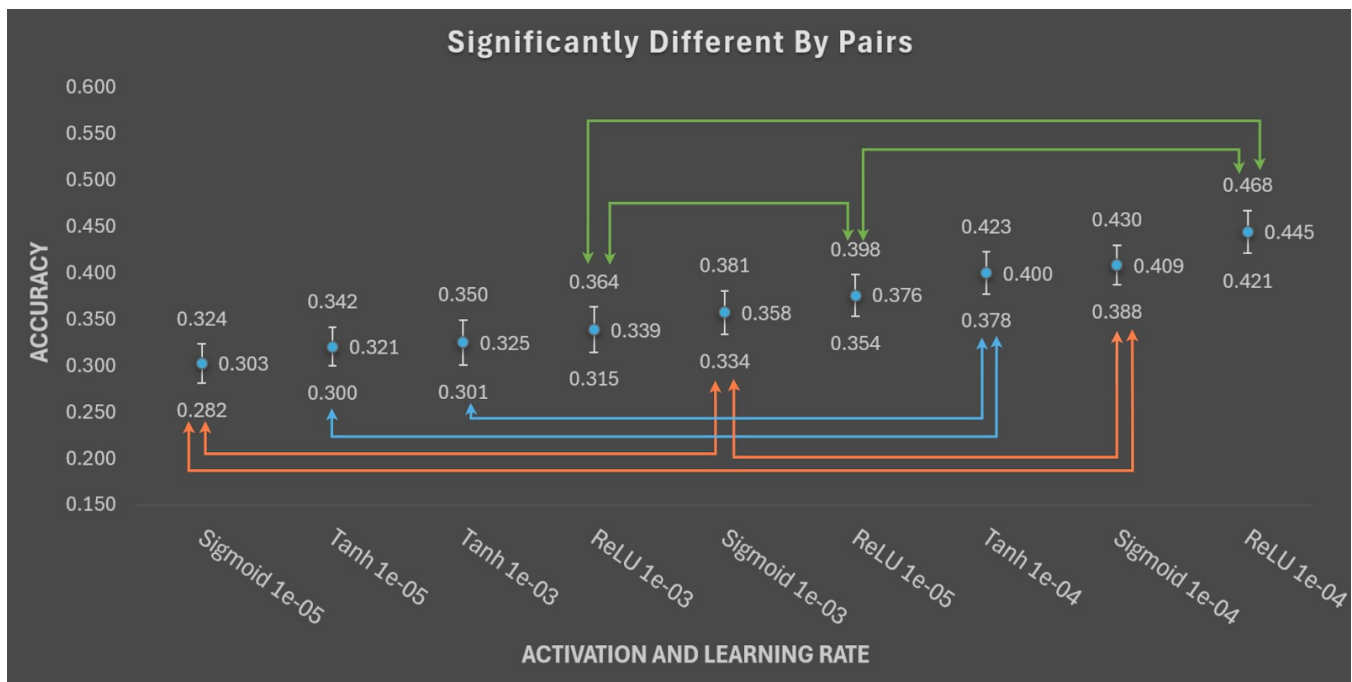


Figure 13: CNN model accuracy results with 95% confidence intervals with lines representing a statistical difference between accuracy.

10. References

- [1] I. Bendib, A. Gattal, and G. Marouane, "ISPR '20: The international conference on Intelligent systems and Pattern recognition," in *ISPR '20: Proceedings of the 1st International Conference on Intelligent Systems and Pattern Recognition*, 2020, pp. 67–70
- [2] A. Bombini, "Using CNN to classify images w/PyTorch," Kaggle, <https://www.kaggle.com/code/androbomb/using-cnn-to-classify-images-w-pytorch> (accessed Feb. 13, 2024).
- [3] P. Budhwant, "A beginner's guide to deriving and implementing backpropagation," Medium, <https://medium.com/binaryandmore/beginners-guide-to-deriving-and-implementing-backpropagation-e3c1a5a1e536> (accessed Feb. 7, 2024).
- [4] Y. Chen, "ESSE 2020: 2020 European Symposium on Software Engineering," in *ESSE '20: Proceedings of the 2020 European Symposium on Software Engineering*, 2020, pp. 137–141
- [5] N. Cibiraj and M. Varatharajan, "Chattering reduction in sliding mode control of quadcopters using neural networks," *Energy Procedia*, vol. 117, pp. 885–892, Jun. 2017. doi:10.1016/j.egypro.2017.05.207
- [6] A. Hossain and S. Sajib, "Classification of Image using Convolutional Neural Network (CNN)," *Global Journal of Computer Science and Technology: D Neural & Artificial Intelligence*, vol. 19, no. 2, 2019. Accessed: 2024. [Online]. Available: <https://doi.org/10.34257/GJCSTDVOL19IS2PG13>
- [7] D. Johnson, "CNN Image Classification in TensorFlow with Steps & Examples," Guru99, <https://www.guru99.com/convnet-tensorflow-image-classification.html> (accessed 2024).
- [8] K. Kalra, "Convolutional Neural Networks for Image Classification," Medium, <https://medium.com/@khwabkalra1/convolutional-neural-networks-for-image-classification-f0754f7b94aa> (accessed 2024).
- [9] J. Le, "The 4 Convolutional Neural Network Models That Can Classify Your Fashion Images," Medium, <https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d> (accessed 2024).
- [10] L. Liu, "ICITEE2021: The 4th International Conference on Information Technologies and Electrical Engineering," in *ICITEE '21: Proceedings of the 4th International Conference on Information Technologies and Electrical Engineering*, 2021, pp. 1–5
- [11] J. McDermott, "Convolutional Neural Networks - Image Classification w. Keras," Learn Data Science, <https://www.learndatasci.com/tutorials/convolutional-neural-networks-image-classification/> (accessed 2024).
- [12] M. Mishra, "Convolutional Neural Networks, Explained," Medium, <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939> (accessed 2024).
- [13] D. Mwititi, "Image Classification with Convolutional Neural Networks (CNNs)," KDnuggets, <https://www.kdnuggets.com/2022/05/image-classification-convolutional-neural-networks-cnns.html> (accessed 2024).
- [14] G. Panitchakorn and Y. Limpiyakorn, in *ICICSE 2021: 2021 10th International Conference on Internet Computing for Science and Engineering*, 2021, pp. 101–104
- [15] A. Perlato, "CNN and Softmax," AndreaPerlato, <https://www.andreaperlato.com/aipost/cnn-and-softmax/> (accessed 2024).
- [16] G. Sanderson, "Backpropagation Calculus | Chapter 4, Deep Learning," YouTube, <http://www.youtube.com/watch?v=tIeHLnjs5U8&list=PL9QBf8R6+5JSUGFedqNqddSOg-3p8V03CF> (accessed 2024).
- [17] G. Sanderson, "But What Is a Neural Network? | Chapter 1, Deep Learning," YouTube, <http://www.youtube.com/watch?v=aircAruvnKk> (accessed 2024).
- [18] N. Sharma, V. Jain, and A. Mishra, "An Analysis Of Convolutional Neural Networks For Image Classification," *Procedia Computer Science*, vol. 132, pp. 377–384, 2018. doi:10.1016/j.procs.2018.05.198
- [19] H. Shuo and H. Kang, "IPMV 2021: 2021 3rd International Conference on Image Processing and Machine Vision," in *IPMV '21: Proceedings of the 2021 3rd International Conference on Image Processing and Machine Vision*, 2021, pp. 60–65
- [20] A. Singh, "Demystifying the Mathematics Behind Convolutional Neural Networks (CNNs)," Analytics Vidhya, <https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/> (accessed 2024).
- [21] P. Skalski, "Gentle Dive into Math Behind Convolutional Neural Networks," Medium, <https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9> (accessed 2024).
- [22] F. Tariq, "Breaking Down the Mathematics Behind CNN Models: A Comprehensive Guide," Medium, <https://medium.com/@beingfarina/breaking-down-the-mathematics-behind-cnn-models-a-comprehensive-guide-1853aa6b011e> (accessed 2024).
- [23] H. L. Yuan, L. Q. Liu, Y. Chen, and F. C. Zhang, "ICIT 2022: IoT and Smart City," in *ICIT '22: Proceedings of the 2022 10th International Conference on Information Technology: IoT and Smart City*, 2023, pp. 118–121