Fall 2023

Object Recognition with Deep Neural Networks in Low-End Systems

Lillian Davis
Western Kentucky University, lillian.davis956@topper.wku.edu

OBJECT RECOGNITION WITH DEEP NEURAL NETWORKS IN LOW-END SYSTEMS

A Capstone Experience/Thesis Project Presented in Partial Fulfillment

of the Requirements for the Degree Bachelor of Science

with Mahurin Honors College Graduate Distinction

at Western Kentucky University

By

Lillian Davis

November 2023

*****

ABSTRACT

Object recognition is an important area in computer vision. Object recognition has been advanced significantly by deep learning that unifies feature extraction and classification. In general, deep neural networks, such as Convolution Neural Networks (CNNs), are trained in high-performance systems. Aiming to extend the reach of deep learning to personal computing, I propose a study of deep learning-based object recognition in low-end systems, such as laptops. This research includes how differing layer configurations and hyperparameter values used in CNNs can either create or resolve the issue of overfitting and affect final accuracy levels of object recognition systems. The main contribution of this thesis research is an evaluation of various approaches in structuring and training deep learning neural network object recognition algorithms. The experiment discovers what patterns exist in the hyperparameters and layering designs to achieve high performance under limited computational resources.

I dedicate this thesis to my parents, Brandy and John, who have loved and supported me in everything I do. I also dedicate this to my friends, Abby and Ethan, whom I am so lucky to have met, and who have kept me sane throughout college.

ACKNOWLEDGEMENTS

I am thankful for Dr. Li, my first reader and academic advisor, who greatly helped me throughout my research process. Thank you for being patient, teaching me, and answering all my questions. I am also grateful for Dr. Xing who has assisted me in my academic and professional career. Thank you for going out of your way to help me and for all the letters of recommendation.

A special thanks to WKU and the Mahurin Honors College for giving me a place to study and call home for the past three and a half years. Thank you for providing such a welcoming institution where I have made some great memories and lifelong friendships.

I am thankful for my parents, Brandy and John, for always encouraging and supporting me, both emotionally and financially. Thank you to Walker for being the best big brother I could ask for. Also, I would like to thank my Granny who is wonderful and kind in everything she does. Thank you all for being such a loving and supportive family.

I would like to thank my best friends, Tori, Holly, Abby, and Ethan. Thank you for the emotional support and for listening to me ramble about computer science, even when you didn’t understand a word I was saying.

Finally, I would like to thank my friends, family, and all those who supported me and provided encouragement.

VITA

EDUCATION

Western Kentucky University, Bowling Green, KY	Dec. 2023
B.A. in Computer Science – Mahurin Honors College Graduate	
Honors Capstone/Thesis: *Object Recognition with Deep Neural Networks in Low-End Systems*	
Daviess County High School, Owensboro, KY	May 2020

PROFESSIONAL EXPERIENCE

Kentucky Office of Homeland Security Web Development Internship	Jan. 2022 – June 2022
Jasper Industrial Supply, Application Developer Intern	May 2022 – May 2023
Jasper Industrial Supply, Integration Architect	May 2023 – Aug. 2023
The Gatton Academy of Mathematics and Science, WKU	Sept. 2021 –
Computer Science Tutor	Dec. 2023

AWARDS & HONORS

Summa Cum Laude, WKU	Dec. 2023
Fruit of the Loom Outstanding Undergraduate Student in Computer Science	May 2022
Outstanding Junior in Computer Science	May 2021

CONTENTS

LIST OF FIGURES

*All diagrams were made using Lucid Chart and Google Drawings

INTRODUCTION

As artificial intelligence becomes increasingly common in the real-world, people are naturally drawn to explore the capabilities of advanced machinery. One of these exciting advancements comes in the form of Computer Vision. Computer Vision is described as a branch of artificial intelligence that helps a computer to “see” like a human, allowing it to understand and properly react to learnt behavior. The goal of computer vision is to programmatically take in information from visual sources and accumulate data to make inferences or perform actions based on the results [21].

Object recognition stands as a paramount element in the domain of computer vision. When presented with an image, recognition algorithms are designed to identify a category to which the object belongs. The advancement of object recognition has significantly progressed through the adoption of deep learning that unifies feature extraction and classification processes. Deep learning-based object recognition has demonstrated exceptional classification accuracy, especially when equipped with large training datasets and neural networks of deep layers. These networks demand substantial computational power and high memory capacity, usually fulfilled by high-performance systems leveraging powerful processors and extensive memory resources [14]. However, few works have been done to explore the performance of deep learning systems with limited computing resources in the context of object recognition.

In this thesis, I aim to investigate the functionality of a deep neural network operating within a low-end system, such as a laptop. The motivation behind this research is to expand the application of deep learning to personal computing. My focus will be on examining Convolutional Neural Networks (CNNs) composed of numerous convolution layers for feature extraction and multiple fully connected layers for classification. It is essential to highlight that CNNs differ from traditional classification methods, as their convolution filters are learnt from the training data, as opposed to being pre-designed by experts for a specific classification task.

Specifically, this thesis will examine the accuracy levels through an experiment of creating a neural network with a public training dataset in a low-end system, to determine the effectiveness of various approaches in structuring and training object recognition algorithms. This experiment attempts to find the best hyperparameters needed to achieve the highest accuracy rate. The experiment results show the differences between using a max or average pooling layer and balancing the quantity of training data with the number of system layers. From the results, we can conclude that a max pooling layer outperforms an average pooling layer. The training data set and number of layers in the CNN must be balanced to best classify and predict data. Additionally, there needs to be a larger amount of feature maps created in the feature extraction phase and number of neurons fired in the classification phase when training the system with more complex or detailed visual data.

There are many challenges hindering the achievement of high-accuracy rates in computer vision systems. This thesis will focus on two primary obstacles found in many computer vision systems that affect the overall quality of object detection and recognition systems. The first issue being a lack of training data, as seen in some systems utilized in automated vehicles. The second issue will observe obstacles caused by ineffective hyperparameter values, such as an unbalanced correlation between the quantity of visual data in training data sets versus the number of system layers, that could result in overfitting.

WHAT IS COMPUTER VISION?

Before delving into the accuracy of final descriptions gained from computer vision systems, it is important to understand the process behind building a system to best obtain and read these results. Therefore, we ask the question: what is computer vision and how is it being used?

Computer vision is a branch of Artificial Intelligence (AI). In this branch of AI, systems using computer vision will programmatically read in information from images, videos, and other visual representations, to accumulate data and make inferences or perform actions based on the received data [20, 6]. Computer Vision helps the computer to “see” human behavior, allowing it to understand and properly react to the learnt behavior [21].

One of the main goals of computer vision is to try to match or even surpass the average human visual capability [22]. These visual capabilities include telling distance between objects, motion or direction of objects, recognizing and distinguishing between objects, and determining if something is wrong or if an object contains any flaws [21, 7]. Human beings already have these capabilities, now the goal is for computers to do the same. To achieve these lofty aspirations, software developers and AI specialists have been researching and experimenting with computer vision systems since as early as the 1950s [20, 3].

There are three main steps in computer vision systems: gaining visual data, analyzing the given data, and reacting to the compiled results [20]. With today’s technology, computer vision systems can rely on various aspects such as sensors, cameras, or visual databases to acquire visual input. The system can communicate with advanced mechanical devices, such as monitors, cell phones, or even vehicles to carry out the appropriate responses to the visual data. The question then becomes, between the stages of obtaining visual data and acting on it, how do computers recognize what they are seeing? In other words, how do computers know what they are looking at?

DEEP LEARNING AND CNNS

One of the main techniques computer vision systems use to break down and recognize inputted visual data are CNNs through deep learning [6].

Deep Learning

Machine learning is a technique used by advanced technology to allow computer systems to learn from input and recognize patterns for future data. Machine learning allows computer systems to make predictions, create categories, and perform other calculations on inputted data [3].

There are multiple categories of machine learning with the three main categories being Supervised, Unsupervised, and Semi-Supervised learning. A system's corresponding category is determined by the amount of user interaction needed when training the system to assign the correct categories for the data. Supervised learning requires training data with inputs containing clear labels. After learning the patterns associated with the given data, it can then receive unlabeled data to correctly recognize and categorize. On the other hand, unsupervised learning reads from training data sets without any labels, only looking at the raw data. Semi-Supervised learning is a balance between the two, where the system is given a small set of labeled training data, and then another larger set of unlabeled training data [22]. Deep learning algorithms can be trained using Supervised, Unsupervised, or Semi-Supervised learning. For the scope of this report, we will assume the computer vision system is developed using supervised learning.

Deep learning is a type of machine learning that is used by many computer vision systems. Machine learning requires human interaction to correct any errors the system makes [21]. However, deep learning aims to minimize or eliminate any human interaction by utilizing a neural network to correct any mistakes and create accurate results [22]. Deep learning is typically used in high-end systems. These systems include high-speed GPU and a larger storage in main memory. As a result, the systems can handle larger computational tasks than low-end systems.

Hyperparameters are an important aspect in deep learning algorithms. Hyperparameters are set before the system is trained. These are parameters that can be manually changed to best optimize the system and receive the greatest accuracy rate. There are multiple hyperparameters including the number of nodes in each fully connected layer, number of final classification categories, learning rate, number of layers, training data size, decision of pooling strategy (maximum vs average vs minimum pooling), filter sizes and more. Fluctuations in any of these values can affect the way in which the object recognition system breaks down, filters out, and classifies objects through a CNN.

When implementing a deep learning system, advanced algorithms are used to recognize key features in the images based on past training data. The system is given multiple different inputs with annotations or final categorical labels to train the system [22]. The training data should encompass enough data points so that the system contains the necessary patterns or key recognizable features to correctly categorize or predict future data. Usually, an increased amount of training data used for a system will result in more accurate classifications and predictions. However, this is not guaranteed to be the truth. For example, increased training data and less layers, or a greater number of layers but less data could provide for a weaker or less accurate system.

There are necessary steps in machine learning algorithms taken after obtaining the data. The programmer must choose a model, train the model, evaluate the model, and make any necessary changes so that the system can make predictions for future data.

CNNs

CNNs, or Convolutional Neural Networks, are a part of deep learning. CNNs are used to break down an image into a matrix of pixels. Each point in the matrix is given a label based on the pixel RGB count: 0 (black) - 255 (white). In a CNN, processing an image is based on the studying of these matrices and determining a pixel's relation to its neighbors [3].

The CNN first uses edge detection to determine edges and hard lines before depicting simple shapes; thereby identifying objects and related entities in the image through processes such as object detection and image segmentation [21].

There are various operations and algorithms used to understand the contents of an image including convolutions, pooling, non-linear activations, and more [3]. CNNs perform convolutions to create predictions about the contents in the provided images. These predictions are checked by the neural network and the process is repeated until the predictions are within a close accuracy range. The following sections will provide more in-depth explanations about the construction of CNNs describing some of the capabilities, layering designs, and methods used to increase accuracy within the system.

LAYERS IN A CNN

A neural network is composed of multiple layers, each performing a distinct job to help understand the patterns and recognize objects found in an image. Each layer in a neural network is composed of a series of nodes. There are no links between nodes in the same layer; instead, relationships between one design of the same layer are determined by nodes in the next layer. Each node is specified with a weight. The weights of each node are constantly updated as the process continues. The final output of the CNN is determined by node weights.

The three main types of neural network layers commonly used when designing a computer vision system for object recognition are convolution, pooling, and fully connected layers. A neural network is separated into Preprocessing (feature extraction) and Recognition (classification) layer groups with convolution and pooling making up the preprocessing groups, and fully connected layers being a part of the recognition group [22].

Convolution

The convolution layer is one of the most important layers used for computer vision systems. In many examples of functioning CNN networks this is the first function applied to the given visual data. Convolution integrates the neighboring pixels and can eliminate or blur some detail in the image value. This layer uses small filters (kernels), often set to 3x3 or 5x5 size, to segment the image. In the past, these kernels were manually constructed by the programmer based on similar categorical systems. Nowadays, for more modern systems using deep learning the kernels are constructed during the training process. The filter is originally set to random values, then as the system is trained the kernel values change using the gradient descent of the system to create more accurate results. These kernels are shifted across the image matrix, multiplying the small convolution sections by the filter value to create a convoluted feature [22]. In other words, convolution layers take surrounding pixels and view them as a single group, resulting in a feature map or activity map.

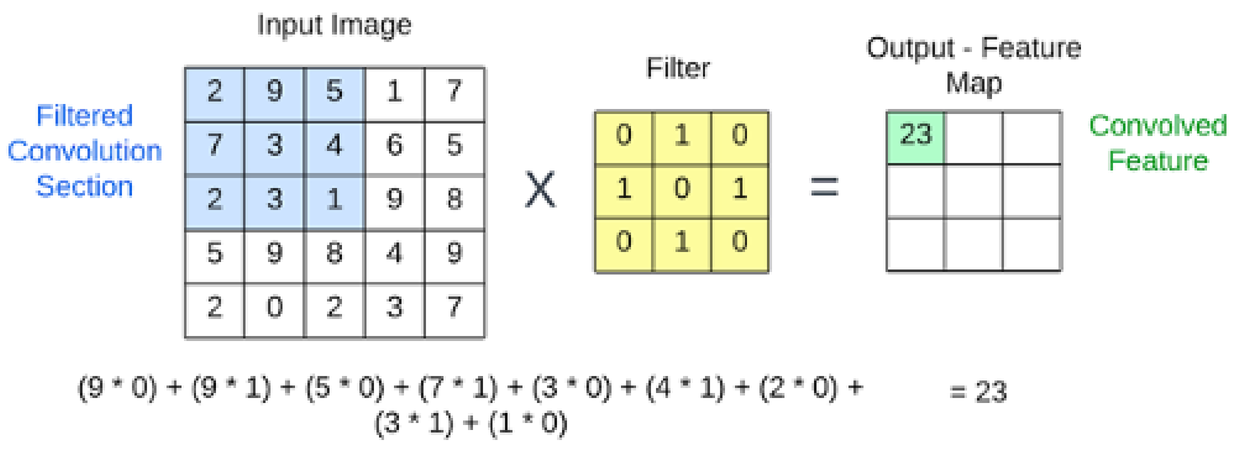


Figure 1: Diagram showing the process for the convolution layer to create a feature map

The ReLU (Rectified Linear Unit) function often occurs after the convolution layer and before the pooling layer. The ReLU is a non-linear activation function commonly used due to its high level of fast convergence. The ReLU affects the feature map created from the previous convolution layer by getting rid of negative values [22]. The function sets negative values to be zero; however, if the value is larger than zero then outputs the raw data. A formula for the ReLU layer is: R = max(0,y) with y being the original value for the selected data [22].

Pooling

A pooling layer is generally added after a convolution layer. In a pooling layer, the network applies another, often smaller, filter to further break down the output in the feature map generated from the previous convolution layers [2]. Depending on which type of pooling layer is required, mathematical operations are performed to calculate the average, minimum, or maximum values found in these smaller filtered versions of the feature map from the previous convolution layer [2]. Pooling layers are used to recognize individual aspects or objects in an image. The resulting feature map is important in distinguishing aspects from their background or surroundings [2].

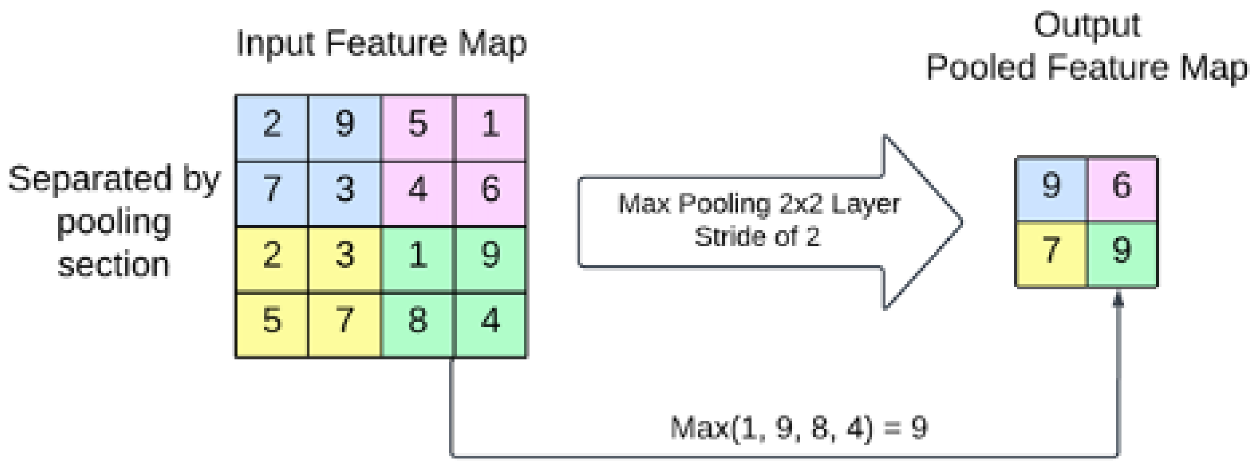


Figure 2: Diagram showing the process for the pooling layer to create a feature map

Both Convolution and Pooling layers are part of the Feature Extraction process. These layers work directly with feature maps created by filtering and extracting smaller segments of the original image.

Up to this point, the layers in the system have been processing the image through a study of matrices. However, the result of the computer vision object recognition process must result in a numerical prediction based on node weights. As a result, programmers introduce the concept of "flattening" at the end of the Feature Extraction process. In this way, the program is transferring the data from the 2-dimensional matrices gained from the feature maps into one-dimensional arrays to be used as the input for the fully connected layers [2].

Fully Connected

The fully connected layer is composed of a series of nodes, or neurons. In a "fully" connected layer, all neurons in one layer connect to all neurons in the previous layer. The connection between these nodes is referred to as the "weight" between the nodes. Typically, these weights are generated randomly with values ranging between 0 and 1. The fully connected layers represent the last layers included in a CNN. There may be more than one fully connected layer found in a CNN. The layers between the original input and final output layers are commonly referred to as hidden layers. Fully connected layers are part of the Classification process, narrowing the final classification down by calculating each node's final weight.

The final output of the CNN will result in a prediction for the determined image category. There are different types of final output layers depending on whether an image can be classified in more than one category. Images that contain one main subject and can be classified into a singular category often utilize a SoftMax layer, where the final weight leads the system to a singular output node [15, 10]. However, visual data that can be conformed to multiple categories require more complex examinations. For these images, the system utilizes multiple logistic regressions to determine a number between 0 and 1 and assign a final category [22].

In a SoftMax layer, each node in the final layer represents an individual image category. Therefore, the SoftMax layer must have an equal number of nodes as the number of available classification categories [10]. The final weights of each node should fall within a range of 0 to 1, with 1 being 100% certain the image should be organized as part of the selected category. All individual weights should sum up to 1 [15, 10]. The classification with a weight closest to 1 will be deemed the final classification [15].

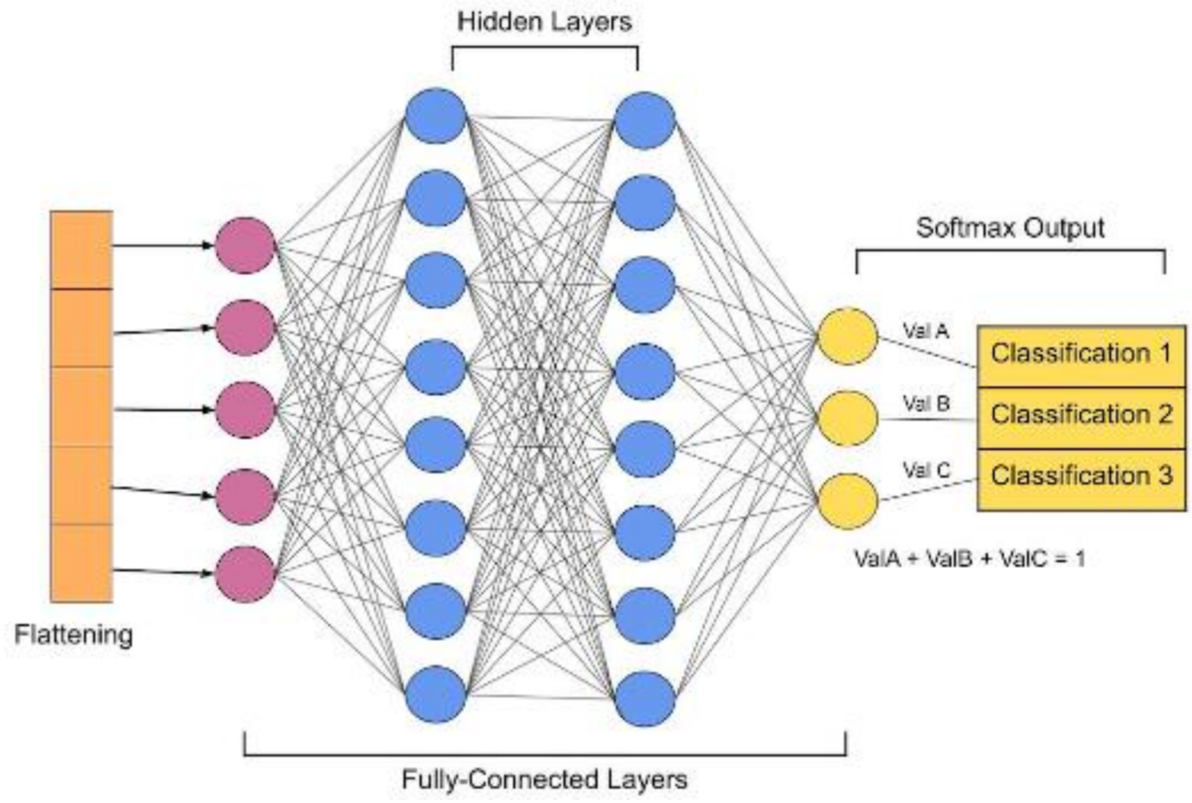


Figure 3: Diagram showing the process for the classification stage through fully connected layers

Each of these layers can be repeated as needed throughout the process. In fact, many systems use multiple convolution, pooling, and fully connected layers to increase the accuracy of the final prediction.

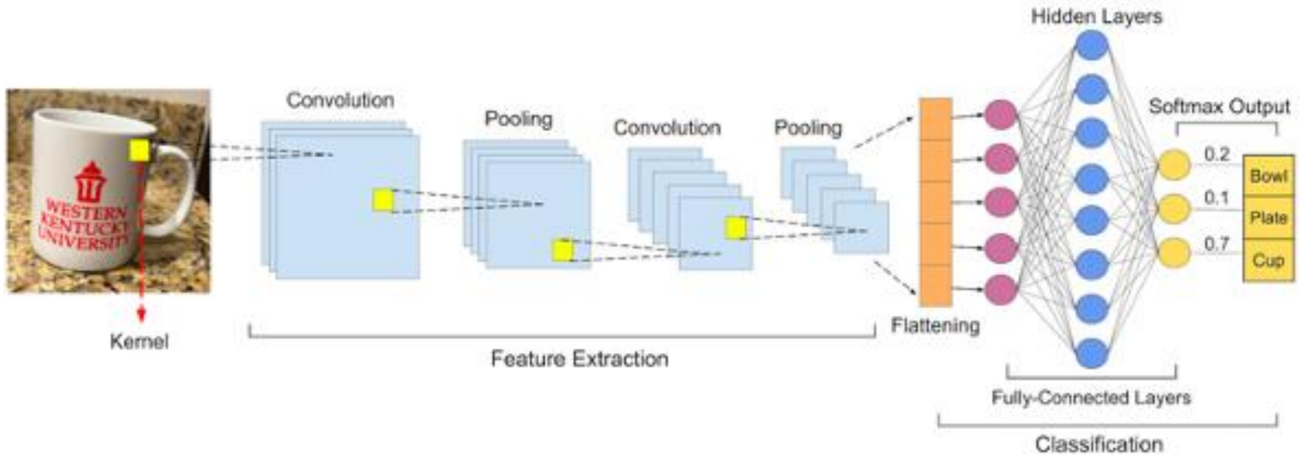


Figure 4: Diagram showing the full Convolutional Neural Network (CNN)

However, this classification prediction may not always be extremely accurate on the network's first try. Therefore, to increase the accuracy of results, programmers have developed ways in which to optimize the system.

OPTIMIZING THE RESULTS

A neural network is essentially a long, somewhat complicated, function. The training process of a neural network is the optimization of all the parameters of the function. Optimization is not used to find the final classification, but rather to find the best parameters to use for the data set [16]. In this way, as the neural network function continues, the weight of the nodes can be adjusted to better fit a final classification. While there are many ways to optimize a computer vision system, one of the most widely used methods is finding the gradient descent, or minimum cost of the loss function [9].

Gradient descent is an important topic in understanding how this final classification is calculated. Gradient descent is used when training a CNN to help correct any mistakes made throughout the weight calculation processes and help guide the final decision to the correct output node [9]. The error in a CNN is repeatedly calculated after the completion of each layer. The gradient descent calculation returns a number value stating the determined error range based on the node's calculated weight; the layered network will then adjust and update the weight calculations [17]. As a result, once the CNN receives real data, not meant for training, the parameters will have an increased accuracy rate and provide data with a decreased error margin.

Gradient descent is calculated through backpropagation. This means that gradient descent is calculated after the image is processed and the network provides an original "guess" as to the corresponding category [16]. If the output classification is incorrect, the neural network will attempt to correct itself through a process called backpropagation. The system back-propagates through the layers. In this way, the network starts at the final classification node layer and works backwards towards the beginning input layer, recalculating the weight by applying the calculated gradient of descent [16].

Gradient descent is calculated through an equation of derivatives to find the minimum value of the cost function. In more simple terms, gradient descent is searching through the function where the gradient, or slope of a function at any given point, is at a minimum or closest to 0 [17]. Seen below is the function used to find Gradient Descent:

$$p(n+1) = p(n) - N * \nabla f(p(n))$$

$p(n) \rightarrow$ Represents a point on the function

$N \rightarrow$ Represents the system's learning rate

$\nabla f(p(n)) \rightarrow$ The gradient (slope) at a point on the function

$p(n+1) \rightarrow$ The next point, or step, on the function

Utilizing the above formula, the following steps are completed to calculate gradient descent:

1. Initialize the formula by choosing a random starting point on the function (p(n)). This random initialization will also help in the long run by reducing the probability of falling into a saddle-point trap.
2. Use this starting point and solve for ∇f(p(n)).
3. If ∇f(p(n)) is less than or equal to your determined minimum acceptable error range, then stop calculations and return the found parameters for gradient descent. If not, then use ∇f(p(n)) and your learning rate to solve for p(n+1).
4. Repeat steps 2 and 3 until you have either found a point with slope close enough to zero, or reached a specified iteration limit [6, 9, 17].

The user-defined learning rate can greatly impact the final gradient descent calculation. Multiplying the found gradient with the learning rate is used to calculate the "step size" between the determination points [16]. A smaller learning rate can lead to a more effective and specific calculation, while a larger learning rate can negatively impact the calculation by "stepping over" the local minimum point [16]. For instance, the MATLAB code found in Appendix A, Code A calculates gradient descent and plots each determination point as it is used in the function. As seen below, example A has a smaller learning rate of 0.1 while Example B has an increased learning rate of 0.9.

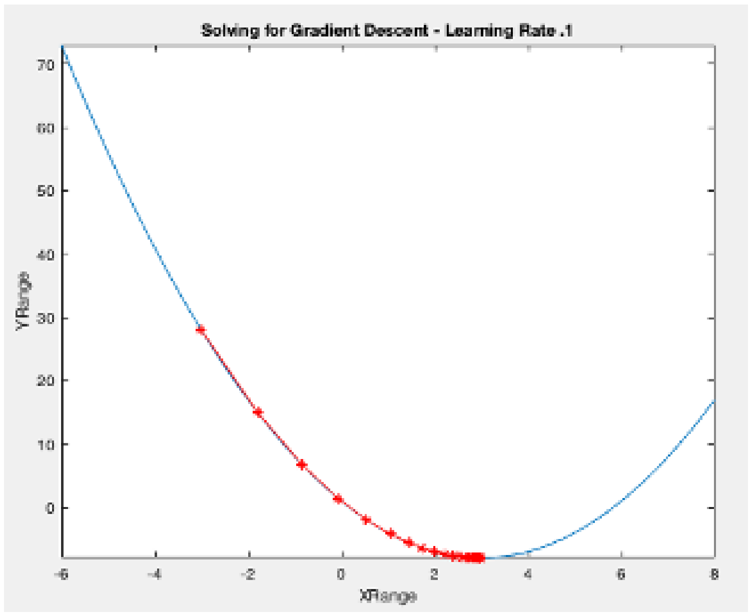


Figure 5: Graph from Code A, all determination points plotted with learning rate of 0.1

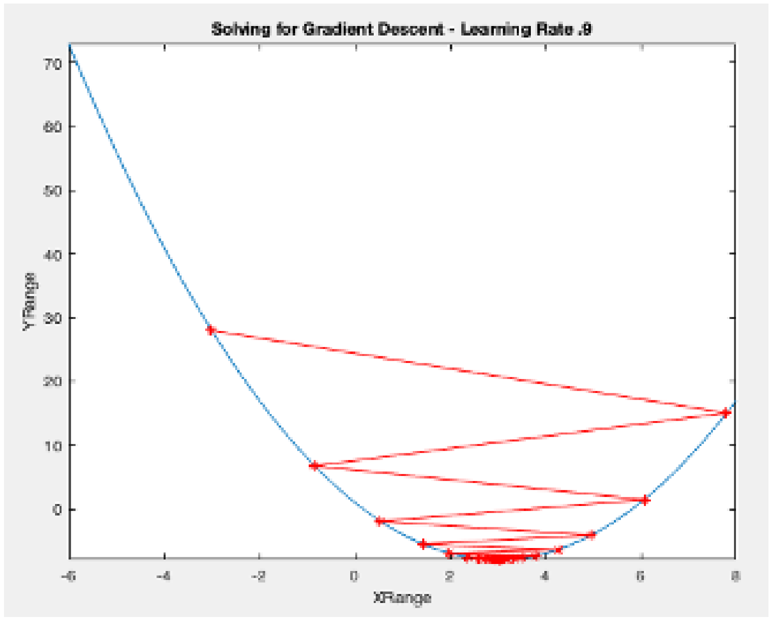


Figure 6: Graph from Code A, all determination points plotted with learning rate of 0.9

Many gradient descent algorithms are charged with finding a local minimum rather than the global minimum. As the CNN becomes more complex, the loss function will also expand with more variables. In multivariable loss functions, the calculation for gradient descent runs the risk of failing to find a global minimum, but instead locating a local minimum [9]. During calculations, the current point could fall into a plateau where the slope is neither increasing nor decreasing in either direction of the current point, instead the graph appears to be flat. Likewise, the algorithm could locate a local minimum in a ridge, where the slope is increasing on both sides. As a result, the system is not guaranteed to find a global minimum. Most of the time this will not drastically affect the system accuracy; however, a few possibilities of escaping this saddle point issue would be to adjust the learning rate or restart the calculations with a different random starting point.

POTENTIAL ISSUES IN OBJECT DETECTION AND RECOGNITION

Although it is an extremely important part of the system, object detection and recognition itself is a major issue in computer vision. There are multiple issues that can affect the overall accuracy results of an object recognition algorithm. To start, object detection can be difficult with clustered backgrounds or poor-quality images. Similarly, pictures taken at angles so the object looks distorted or is hard to recognize can also confuse many object detection and recognition systems [14]. Additionally, computer vision itself can be difficult if businesses lack the machinery or resources necessary for analyzing large amounts of inputted data [4].

Another large issue is a lack of data provided for a “new” object. For example, a system designed to recognize different forms of architecture may be confused by houses built in weird or modern shapes. Another more common example would be found in automated vehicles. There have been many instances where a self-driving car does not have enough training data to determine a classification for unusual objects such as a horse-drawn carriage or pedestrian walking beside a bicycle [11].

Furthermore, overfitting and underfitting are two large issues that must be addressed in computer vision machine learning systems. Overfitting refers to configuring the hyperparameters of the function so closely to the training data that it cannot account for extra or new data. On the other hand, underfitting of the system fails to recognize patterns within the learning data and sets hyperparameters too loose to truly understand the data. Both issues can lead to inaccurate predictions for future data. To mitigate the issue of overfitting and underfitting, it is important to pay close attention to the system's hyperparameters and attempt to configure these hyperparameters wisely.

While many of these issues may not seem extremely advanced or dangerous to the human eye, these conditions can lead to major differences in determination in computer vision systems. Depending on the system's purpose, these differences can lead to devastating and potentially life-ending results. To further demonstrate the tremendous impact these systems can have on everyday life, a person should study the recent achievements and setbacks found in modern computer vision systems such as the ones incorporated in automated vehicles.

REAL LIFE USAGES

Currently, these technologies and neural networks are actively working in almost every aspect of daily life. Computer vision is used in daily activities such as using facial recognition systems and image restoration programs. Retail stores are now using robots equipped with AI object detection systems to quickly count inventory or restock products. Computer vision is also used widely in manufacturing and construction companies wherein image segmentation is used during safety inspections to find any flaws in production. Augmented reality has also become increasingly popular among the public, object recognition systems are used within these high-definition systems to construct and recognize real-time object data in the environment around a person.

Additionally, CNNs are used in more dire situations such as with medical imaging and video surveillance systems. For medical imaging, the machine uses image segmentation on X-rays or other medical images to find different diseases, tumors, and other harmful issues in the image. Intelligent video surveillance and analysis systems use image segmentation to identify key objects, such as humans, in the visual data. These systems can also incorporate facial recognition to later identify people in these captured instances. A market analysis report conducted in 2022 determined the Global Computer Vision Market was valued at about $14.10 billion. As computer vision grows in popularity, this number is predicted to also expand to reach a compound annual growth rate of about 19.6% within the next 7 years [4].

In current times, computer scientists and AI specialists are more closely examining the impact of computer vision capabilities specifically in the transportation industry. According to a market analysis report conducted regarding the 2022 market, 49% share of the Global Computer Vision Market was computer vision systems designed for industrial purposes. While this category does include other manufacturing businesses such as pharmaceutical and electronics development; this category was largely composed of automotive and transportation industries [4]. AI research and understanding has rapidly expanded in recent years, allowing companies such as Tesla and Waymo to attempt to develop fully functioning self-driving cars. The end goal of these autonomous vehicles is creating an increased accessibility and ease for the consumer along with decreasing roadway hazards [4]. However, these cars are not currently filling the market as there are still many potential hazards the car companies must fully solve before they are safe for the masses. This thesis will more closely examine the CNNs used when designing and developing the AI systems in these independent vehicles.

AUTOMATED VEHICLES

The Society of Automotive Engineers (SAE) created five different levels of automaticity for self-driving cars. Currently, modern technology limits self-driving car designs to be categorized as level two, with the driver still maintaining control over necessary functions such as steering and braking. However, the goal for self-driving cars is to be able to be categorized as level five, giving complete control to an AI system driving the vehicle so that there is little to no need for human interaction [13].

An additional goal for AI driven vehicles, other than human convenience, is to improve roadways by lowering traffic rates and cut down on accidents caused by distracted driving. According to Abhishek Balasubramaniam and Sudeep Pasricha following a NHTSA 2021 report, “more than 36,000 people died in 2019 due to fatal accidents on U.S. roadways”. Of these accidents, about 94% were caused by distracted driving or other human errors [13].

Software developers and engineers intend to achieve these goals of automated driving and lowering roadway hazards by utilizing high-detection sensors, including cameras, radars, and lidars, to take real-time snapshots of the vehicle’s surroundings. The AI computer vision software would then classify any objects or roadway hazards in the snapshots and use this information to determine the best course of action to avoid any accidents or mistakes on the road.

However, car designs today are still a long way away from reaching these goals. According to statistics gained from a medical and legal referral company, “Around 9.1 driverless car crashes occur per million miles driven…Over 20 months, Waymo’s self-driving cars have been involved in 18 accidents…In the past four years, eleven Tesla self-driving vehicle accidents have been reported…Overall, there have been around 37 Uber test vehicle crashes” [11]. This shows that there are definitive issues that still need to be addressed before humans can take a fully hands-off approach on the road.

Among these issues include a lack of trust in many consumers. A survey taken by the Boston Consulting Group in 2016 determined that the number one concern people have with self-driving cars is that they would not feel safe in the vehicle; 50% of all respondents agreed with this statement. People do not feel willing to get in a self-driving car while others do not even trust being on the road with AI driven machines [11]. Many consumers want to always maintain control of their vehicles in case of emergencies or unsafe roadways.

Another issue arises in the lack of knowledge and statistics concerning the reliability of functions of the cars. While the dream for self-driving cars has been present since as early as the 1930s, the actual development of AI driven cars is still a relatively new feature [11]. Therefore, data is still being gathered about the machines' performances and the idea is still in the development phases.

While these issues cause a lack of headway in the future of self-driving cars; one of the most prevalent and deadly issues with the technology is the lack of certainty in object detection and recognition systems. This decreased certainty can stem from many places; one of the most common causes is a lack of training data. An extremely large quantity of training images is necessary to properly train a system as complex as automated vehicles. One instance of automated systems was trained utilizing a DCNN with a training dataset of over one million records to avoid overfitting [18]. However, even this amount of data cannot always account for new or entirely unexpected obstacles that often appear while driving. Humans have the capability of seeing an object on a road and, without even fully knowing what the obstacle is, making quick decisions about whether to go around the object, stop the car, or continue as normal. Currently, some companies struggle to reach the same goal in object detection and recognition systems.

For instance, the tragic death of Elaine Herzberg on May 7, 2018, was the first recorded pedestrian fatality at the fault of a malfunction in an AI driven vehicle. Herzberg was walking alongside her bike and crossing the road when she was struck by an AI driven Volvo XC 90. It was later found out that the object recognition system lacked substantial data to truly determine what the upcoming pedestrian walking next to a bicycle should be classified as. Herzberg was recognized by the AI software as an unknown object, car, and bicycle within the span of 6 seconds [11]. Sadly, the AI did not make a reasonable decision to stop before colliding with the unexpecting pedestrian. This unfortunate incident displayed early the lack of data, software, and technology needed to successfully fulfill the object recognition capabilities of self-driving vehicles.

This lack of data is not a solitary incident. Other systems have reported issues in identifying unknown objects such as road signs with attached stickers and even horse drawn carriages on roadways. Researchers have reported that more data should be obtained for dealing with changes in weather conditions, terrain, traffic, and location [13]. Training the system using semi-supervised learning or an open dataset can also account for new data such as changes in weather or light [13].

EXPERIMENTS

As previously mentioned, there are multiple attributes that could affect the system’s final classification results of an image. For instance, image quality, object orientation, and extraneous noise could all potentially disrupt the accuracy of the outputs. However, for this thesis, I chose to focus more directly on elements regarding the setup of the computer vision system itself that could affect the accuracy of the results in low-end systems; more specifically, how changing the hyperparameters of type of pooling layer and number of hidden layers regarding the size of the training data set will affect the final output and accuracy of the system.

Hypothesis 1: The training data size and number of layers are connected in providing accurate classification results. A larger training data set with fewer layers will produce a decreased accuracy through underfitting, while a smaller training data set with an overabundance/immense number of layers will produce a decreased accuracy rate through overfitting.

Hypothesis 2: Using a max pooling layer instead of an average pooling layer will result in a higher accuracy rate.

To test these hypotheses, I have created a computer vision system using the MATLAB programming language. The following sections will go further into detail about the process of setting up and implementing the program, obtaining training data sets, running multiple tests, thoroughly examining the obtained results against the original hypothesis, and examining any corrections or prevalent errors found within the program or additional matters that affect the compilation and output of the program.

Before beginning, since this program mainly deals with visual data, it is important to understand how these imported images are perceived in the code. With the MATLAB language, the images are broken down into matrices of values, with each individual value being a numerical digit representative of the corresponding pixel in the image. Each matrix contains certain attributes for the image including dimension size and RGB value. Below is a segment of code used to load a relatively simple, small image into the program.

```
img = imread("Western_Cup.png");

imshow(img);
```

The program reads in the image as a matrix value and displays the image on-screen. Seen below is a visual representation of the image more familiar to what a human would recognize and the corresponding matrix of values that is used by the program to understand the image.

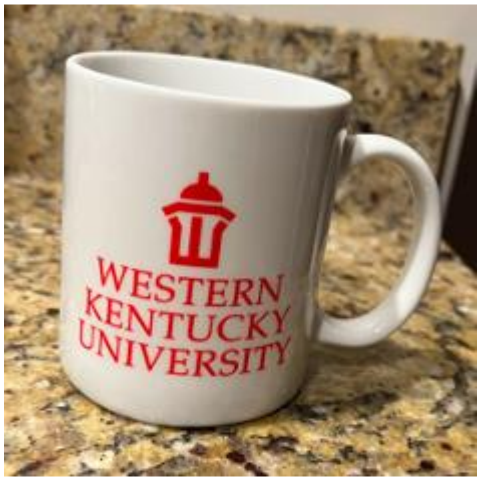
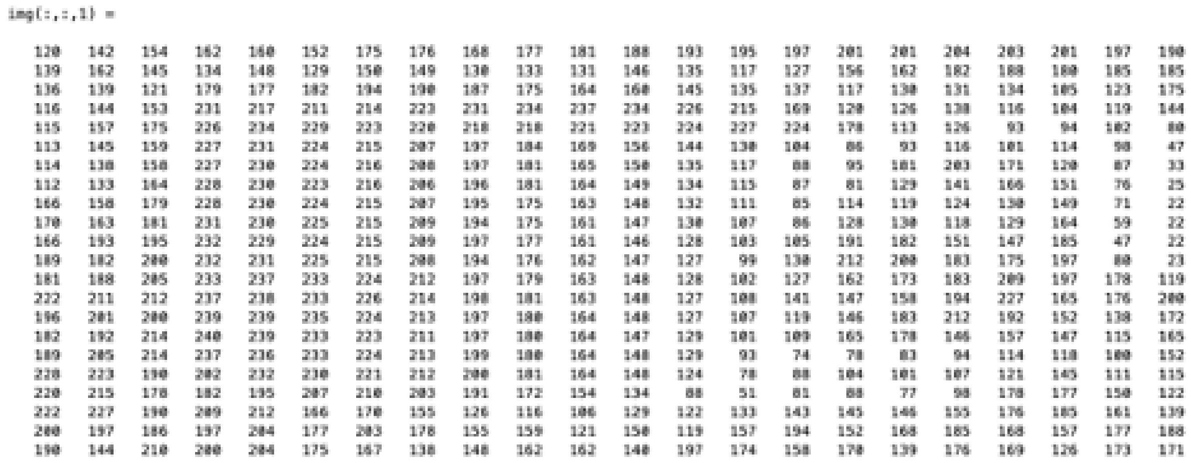


Figure 7: Image of a mug



```
img(:,:,1) =

   120   142   154   162   160   152   175   176   168   177   181   188   193   195   197   201   201   204   203   201   197   190
   139   162   145   134   148   129   150   149   130   133   131   146   135   117   127   156   162   182   188   180   185   185
   136   139   121   179   177   182   194   190   187   175   164   160   145   135   137   117   130   131   134   105   123   175
   116   144   153   231   217   211   214   223   231   234   237   234   226   215   169   120   126   130   116   104   119   144
   115   157   175   226   234   229   223   220   218   218   221   223   224   227   224   170   113   126    93    94   102    80
   113   145   159   227   231   224   215   207   197   184   169   156   144   130   104    86    93   116   101   114    98    47
   114   130   150   227   230   224   216   208   197   181   165   150   135   117    88    95   101   203   171   120    87    33
   112   133   164   228   230   223   216   206   196   181   164   149   134   115    87    81   129   141   166   151    76    25
   166   150   179   228   230   224   215   207   195   175   163   148   132   111    85   114   119   124   130   149    71    22
   170   163   181   231   230   225   215   209   194   175   161   147   130   107    86   128   130   118   129   164    59    22
   166   193   195   232   229   224   215   209   197   177   161   146   128   103   105   191   182   151   147   185    47    22
   189   182   200   232   231   225   215   208   194   176   162   147   127    99   130   212   200   183   175   197    80    23
   181   188   205   233   237   233   224   212   197   179   163   148   128   102   127   162   173   183   209   197   178   119
   222   211   212   237   238   233   226   214   198   181   163   148   127   108   141   147   158   194   227   165   176   200
   196   201   200   239   239   235   224   213   197   180   164   148   127   107   119   146   183   212   192   152   138   172
   182   192   214   240   239   233   223   211   197   180   164   147   129   101   109   165   178   146   157   147   115   165
   189   205   214   237   236   233   224   213   199   180   164   148   129    93    74    78    83    94   114   118   100   152
   228   223   190   202   232   230   221   212   200   181   164   148   124    78    88   104   101   107   121   145   111   115
   220   215   178   182   195   207   210   203   191   172   154   134    88    51    81    88    77    98   178   177   150   122
   222   227   190   209   212   166   170   155   126   116   106   129   122   133   143   145   146   155   176   185   161   139
   200   197   186   197   204   177   203   178   155   159   121   150   119   157   194   152   168   185   168   157   177   188
   190   144   210   200   204   175   167   138   148   162   162   140   197   174   158   170   139   176   169   126   173   171
```

Figure 8: Section of MATLAB data after reading in the image of the mug

Now that it is clear how the MATLAB program reads in visual data, we can begin to explain how this experiment was set up to interpret the objects within said data.

EXPERIMENT SETUP

Available Resources and Limitations

As formerly stated, this experiment is written in the MATLAB programming language. The program incorporates available computer vision, deep learning, and machine learning toolboxes to improve efficiency within the program.

This program was created and run on a low-end system using the resources from a MacBook Air and was thus limited in many ways. The system was trained on a single CPU with 8 GB of RAM. Oftentimes, multiple other programs or resources were running in the background while the system was being trained. To conform to these constraints, the training image datasets were kept relatively small and training time was not allowed to exceed 8 hours. Therefore, this experiment focusses on CNNs running on low-end systems, rather than high-end systems seen in more advanced computations.

Training Image Dataset

One of the most important steps in setting up the experiment was acquiring training data. For the purposes of this experiment, I created three different training datasets from a free public image dataset from Kaggle [20, 8, 1, 5]. Each dataset has a total of 10 different animal categories: cat, dog, squirrel, sheep, horse, elephant, butterfly, spider, chicken, and cow. The difference between the sets is the total number of images in the dataset: 10 images in each category for a total of 100 images, 100 images in each category for a total of 1000 images, and 1000 images in each category for a total of 10000 images. All images in the dataset must be the same size for the input layer. Therefore, after the dataset was read in, all images were resized to an identical

300x300x3 size. Seen below is a small selection of randomly chosen images from the training image dataset.

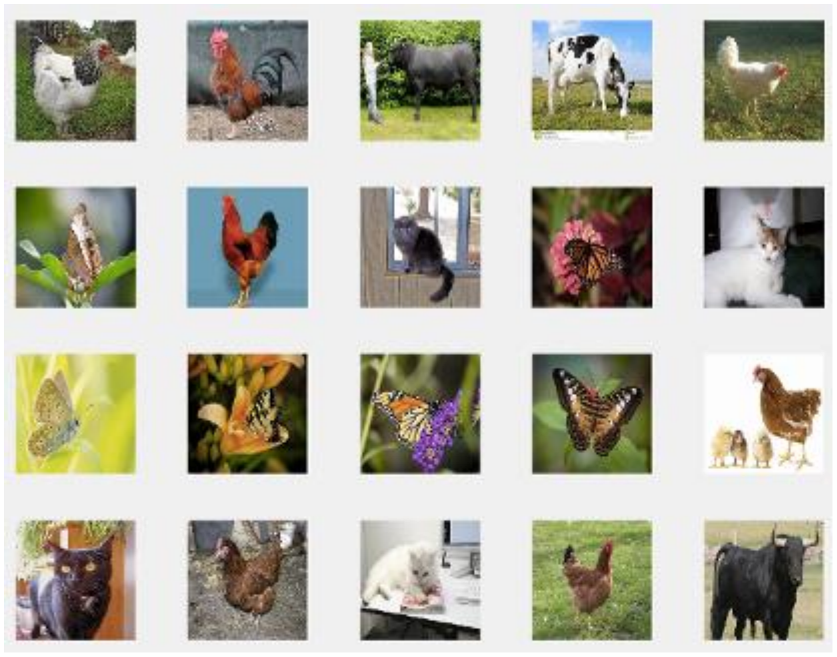


Figure 9: Twenty randomly selected images from the training image dataset

Original Layer Designs

The following section explains the original layering design in the MATLAB program found in Appendix A, Code B for the included layers. The program begins by reading in the local image dataset and resizing the images if needed. The dataset is then split with 70% of the images becoming a training image dataset and the remaining 30% acting as a validation image dataset.

When setting up the original layer designs, the program included an original input layer reading in a matrix of size 300x300x3. The program then contained a convolution 2d layer utilizing a 3x3 filter to create 8 feature maps. The following ReLU layer was used as a non-linear activation function. Afterwards, the feature maps were sent through a max pooling layer with a 2x2 filter and stride of 2 units. This process was repeated, doubling the number of feature maps created in the convolution layer each time.

Next, the feature maps were flattened and transferred to a fully connected layer. The last fully connected layer must have the same number of nodes as final classifications. Therefore, the final fully connected layer contained 10 neurons for the 10 available animal categories. As test runs progressed, these convolutional, ReLU, pooling, and fully connected layers were duplicated as needed and the parameters were changed to provide more precise calculations.

As the system was being trained, the training data accuracy rates were plotted in real time. The figures display two graphs, the training accuracy, and the training loss value, each plotted against the iteration number. In each graph, the system plots the training data (blue line for accuracy, red line for loss) and the validation accuracy (black dotted line). Each point on the validation line is plotted after the completion of a validation minibatch, currently set to a size of 32.

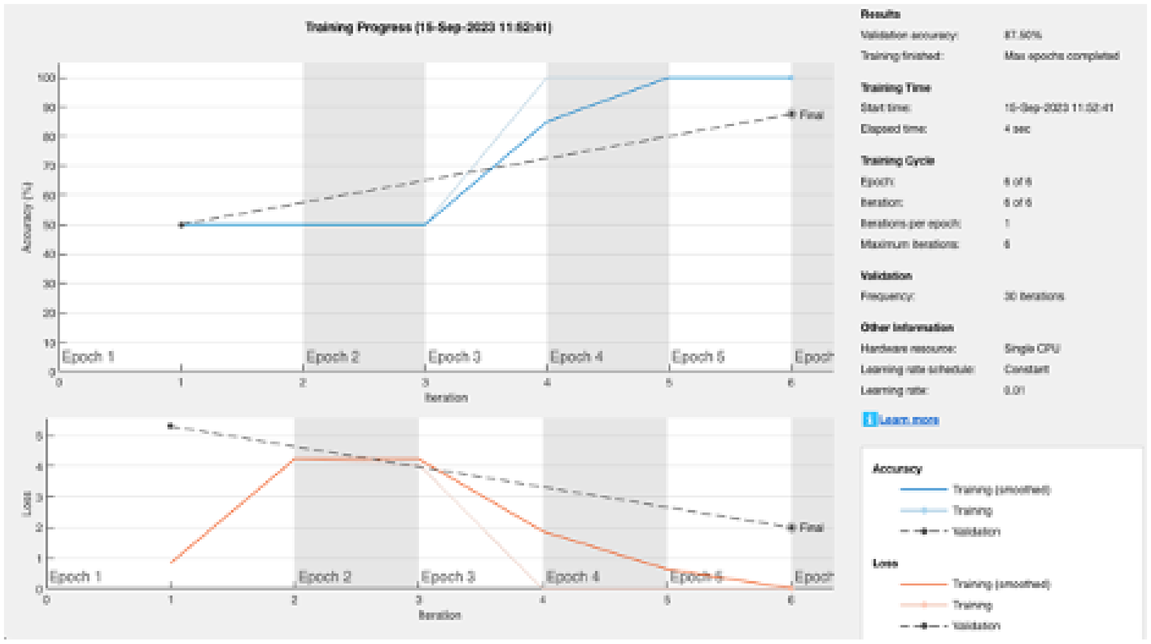
After the CNN is created and trained, the program predicts the categories of the validation image set and terminates after calculating a final accuracy percentage. While there are many ways to calculate the performance of a CNN, such as solving for mean squared error or mean absolute error, this experiment calculates the performance based on prediction accuracy. After training the system, the accuracy is determined by comparing the network's predictions with the actual assigned labels and calculating the percentage of correct classifications.

TEST RUNS

Observations with Small Dataset

To fix any minor errors or bugs in the code without fully training the computer vision system I decided to run the program with a few test runs using an extremely small image dataset. This dataset was compiled of 20 photos on my personal camera roll separated into two categories: cat or dog. Therefore, the training data set consisted of 12 photos with 8 validation images to test for accuracy. The following observations are from test runs performed with this incomplete and minimal image dataset. These runs are solely for understanding the object recognition system and to discover some of the best values to set the hyperparameters for the actual experimental runs. Note that even when running the program with the exact same parameter values multiple times, the algorithm produces different accuracy rates. This is most likely due to the randomization of separating the total image set into training and validation data.

The following observations were all made with an image set size of 20 images. With two sets of convolutional layers the accuracy rate calculated to 75%. When three sets of convolutional layers the accuracy rate is calculated to 50%. This is some indication that an overabundance of layers with a smaller data set could lead to overfitting of the training data. Furthermore, when max epoch is set to 4, the accuracy rate is calculated to 75%; however, when increasing max epoch size to 6 the accuracy rate improved to 87.5%. Since the algorithm ran through the training data set an extra two times it was able to better parameterize the cost function for the system. Originally, the chosen data set was resized so that all images were of the same size 100x100x3. After a few processes of trial-and-error, the images were all reset and resized to the current 300x300x3 dimensions.



Results

Validation accuracy:	87.50%
Training finished:	Max epochs completed

Training Time

Start time:	15-Sep-2023 11:52:41
Elapsed time:	4 sec

Training Cycle

Epoch:	6 of 6
Iteration:	6 of 6
Iterations per epoch:	1
Maximum iterations:	6

Validation

Frequency:	30 iterations

Other Information

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

Learn more

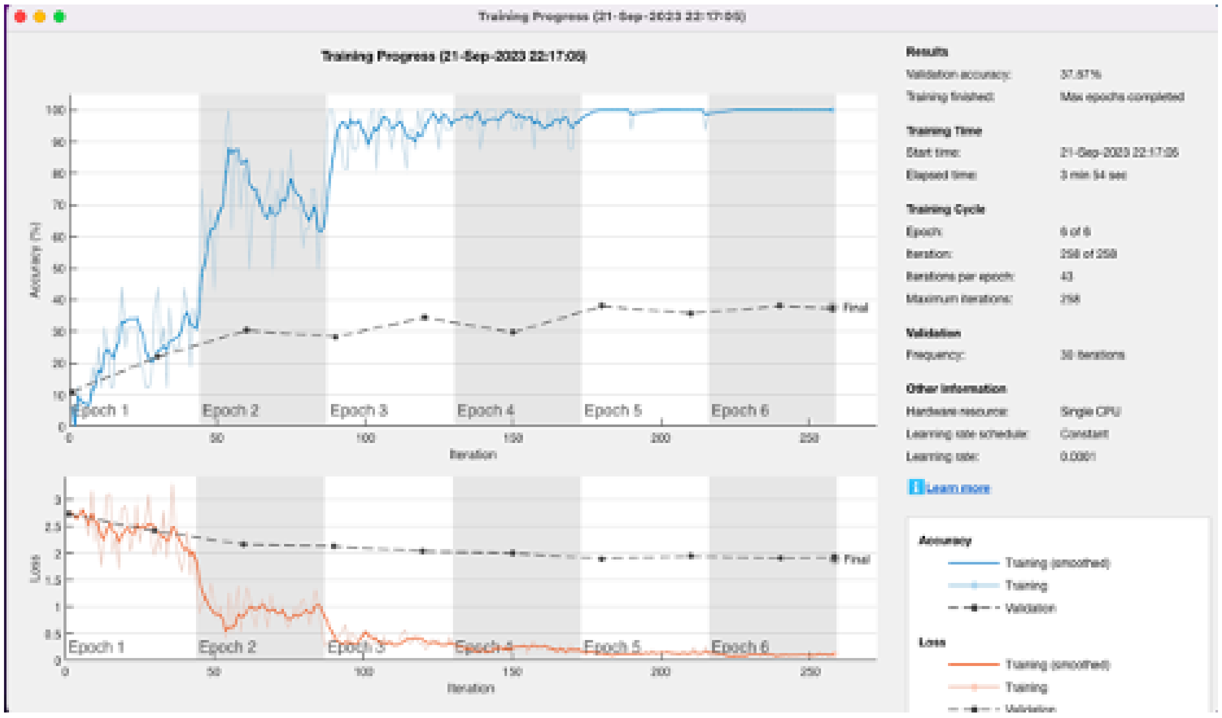
Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Figure 10: Accuracy results from small dataset of 20 images

Examples of Overfitting

The main problem encountered when configuring the parameters of the object recognition system was accounting for overfitting. Seen below is a screenshot of the results for a total image dataset of 1000 images with parameters configured too closely to the original training dataset. This does not allow enough room for new predictions in the validation dataset and results in overfitting.



Training Progress (21-Sep-2023 22:17:05)

Results

Validation accuracy:	37.87%
Training finished:	Max epochs completed

Training Time

Start time:	21-Sep-2023 22:17:05
Elapsed time:	3 min 54 sec

Training Cycle

Epoch:	6 of 6
Iteration:	258 of 258
Iterations per epoch:	43
Maximum iterations:	258

Validation

Frequency:	30 iterations

Other information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Learn more

Accuracy

Training (smoothed)
Training
Validation

Loss

Training (smoothed)
Training
Validation

Figure 11: Accuracy rates depicting overfitting during training

To counter this overfitting issue, the experiment underwent multiple extensive test runs, changing hyperparameters such as learning rate, epoch size, number of features, and the number of convolution, pooling, and fully connected layers. Described below are some of the optimal hyperparameters found and other observations made throughout these test runs.

OBSERVATIONS

Ideal Hyperparameters

The initial learning rate is important in the final accuracy rate; after multiple runs of trial-and-error it is determined that a learning rate of 0.0001 provides the best accuracy results.

The number of epochs determines how many times the system will run through the complete training dataset. In the experiment, for every training dataset, regardless of size, it takes at least 4 epochs for the training dataset to reach near 100% accuracy. The optimal number of epochs is 6. In many test runs, the validation accuracy will level off but still slightly increase in accuracy up until the sixth and final epoch.

The batch normalization layer also helped increase the system's accuracy. Without the batch normalization layer, with a training data set of 20 images, the algorithm was stuck calculating only a 50% accuracy rate; however, after including batch normalization the system improved to a 75% accuracy rate.

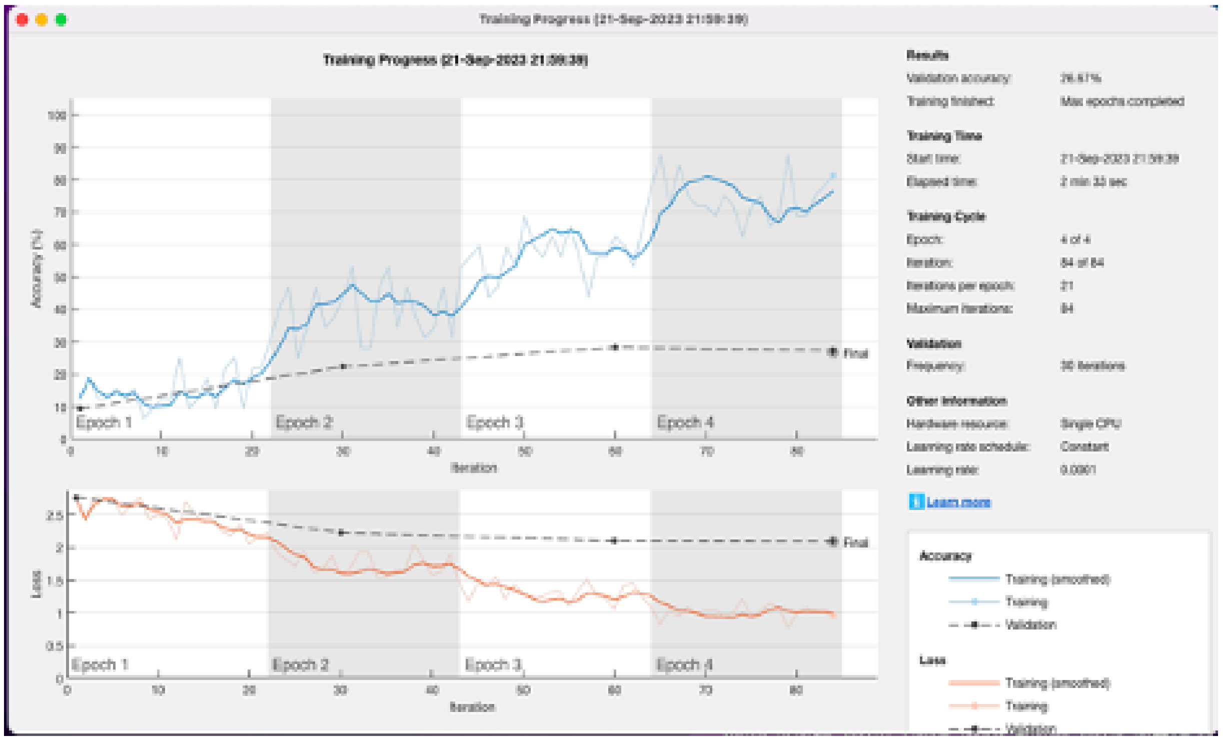
The original images are large in scale and contain an abundance of details, both in the main object of interest and the background or other aspects in the image. Therefore, multiple convolution, pooling, and fully connected layers are required to create feature maps for each of the thousands of features in the image. The program starts with a convolution layer creating 32 feature maps then doubles in size until there are 512 feature maps. There are 6 fully connected layers; the first and fifth layer contain 256 nodes, layers two through four contains 512 nodes, and the final fully connected layer contains 10 nodes matching the number of classifications. There are also two dropout layers after the second and fourth layer to improve accuracy.

Training Time

Another interesting observation was found in the correlation between the training dataset size and the overall training time. The larger the training dataset size, the longer it will take to fully train the system. From the image sets gathered for this experiment, depending on the dataset size, it can take between a couple of seconds to hours to completely train the system. Ideally, a fully operational computer vision system would utilize a training dataset of upwards of 10,000 images. This could take hours or days to fully train the system. However, due to the constraints of this experiment, the training image sets were kept relatively small and therefore took a short time to train. The longest test run in this experiment ran for about 8 hours.

RESULTS

From the multiple test runs, the results of the experiment concluded that the number of convolution, pooling, and hidden layers does influence overall classification accuracy. The original test runs began with a limit of three convolution layers, two max pooling layers, and one fully connected layer. After utilizing the dataset of 1000 total images, the accuracy rates averaged about 20% and took less than 5 minutes to run.



Results

Validation accuracy:	26.67%
Training finished:	Max epochs completed

Training Time

Start time:	21-Sep-2023 21:59:39
Elapsed time:	2 min 33 sec

Training Cycle

Epoch:	4 of 4
Iteration:	84 of 84
Iterations per epoch:	21
Maximum iterations:	84

Validation

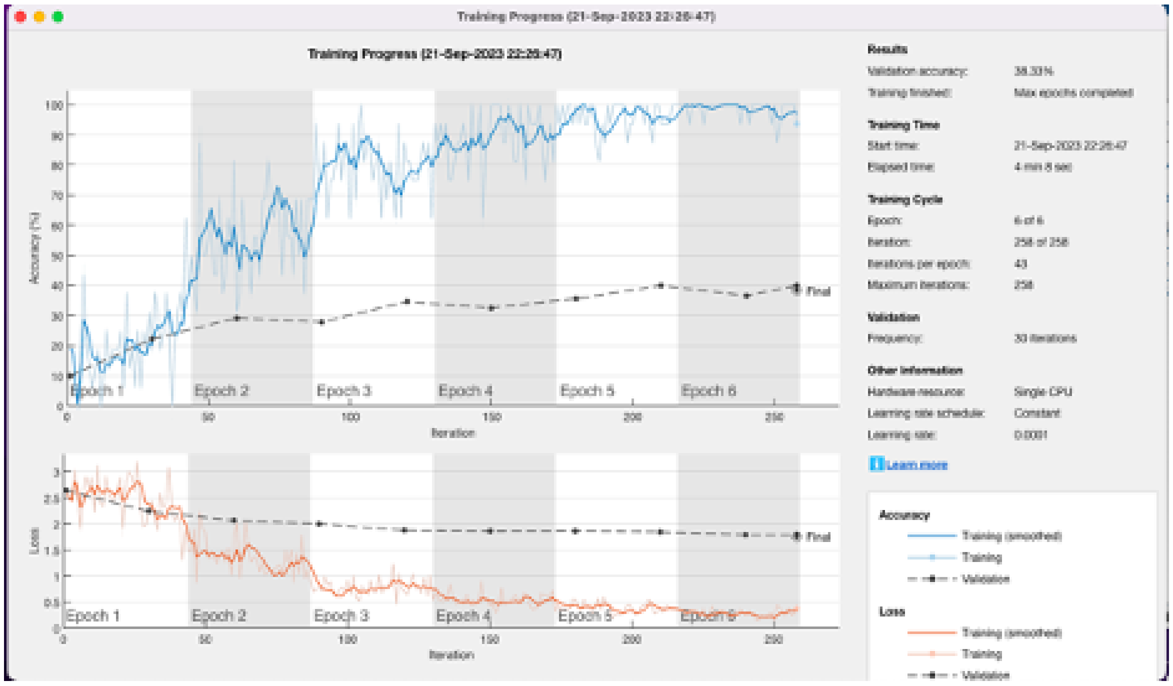
Frequency:	30 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Figure 12: Accuracy rates for initial test run of 1000 images

As the number of convolution and pooling layers increased, the accuracy rates also slightly increased. A major difference came from including multiple hidden, fully connected layers. While this did increase the total elapsed time for training, it also increased accuracy rates by about 10% to 20%.



Results

Validation accuracy:	38.33%
Training finished:	Max epochs completed

Training Time

Start time:	21-Sep-2023 22:26:47
Elapsed time:	4 min 8 sec

Training Cycle

Epoch:	6 of 6
Iteration:	258 of 258
Iterations per epoch:	43
Maximum iterations:	258

Validation

Frequency:	30 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Accuracy
Training (smoothed)
Training
Validation

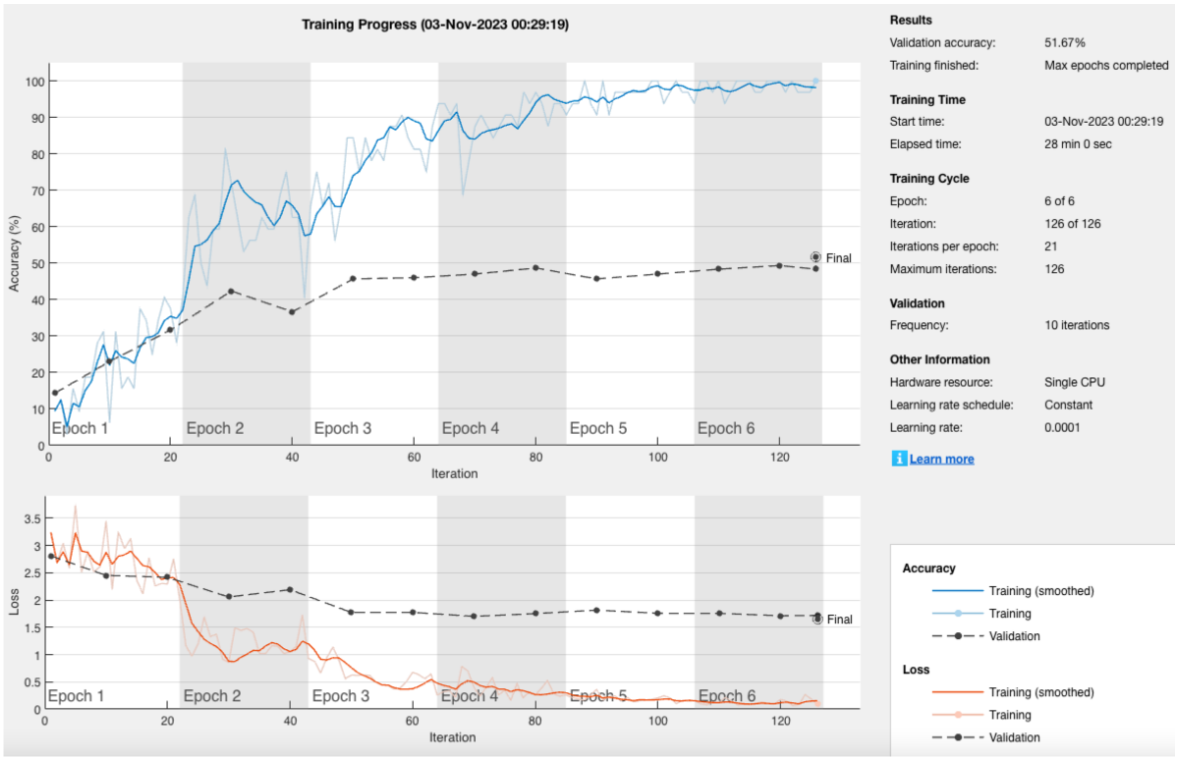
Loss
Training (smoothed)
Training
Validation

Figure 13: Accuracy rates for test run on 1000 images with multiple fully connected layers

However, I discovered that the number of layers was not the only important detail in the layering design. The quantity of features extracted in the convolution layers and the corresponding number of nodes in the fully connected layers have a high correlation to the classification accuracy. This increase in features is due to the large dimensions and number of details found within the image dataset, each image being 300x300x3 in size and containing multiple important features. Since the images are a large size there will be more features in the image. Therefore, the program requires a larger number of filters in the convolution layers. The program starts with a convolution layer with 32 features and doubles until reaching a desired size.

Through multiple trial-and-error test runs; I determined that the current optimal number of features for the convolution layers and number of nodes for the fully connected layers to be between 512 and 1028. Given the fixed size of input, increasing the number of layers will result in long elapsed runtimes, upwards of 8 hours, and a decreased accuracy.

The current highest accuracy was obtained with 4 convolution layers (starting at 32 features and increasing to 512) and 6 fully connected layers (3 layers each with 512 nodes, two layers with 256 nodes, and the last fully connected layer containing 10 nodes). Utilizing an image dataset of 1000 total images, the system resulted in a final validation accuracy of 51.67%.



Results

Validation accuracy:	51.67%
Training finished:	Max epochs completed

Training Time

Start time:	03-Nov-2023 00:29:19
Elapsed time:	28 min 0 sec

Training Cycle

Epoch:	6 of 6
Iteration:	126 of 126
Iterations per epoch:	21
Maximum iterations:	126

Validation

Frequency:	10 iterations

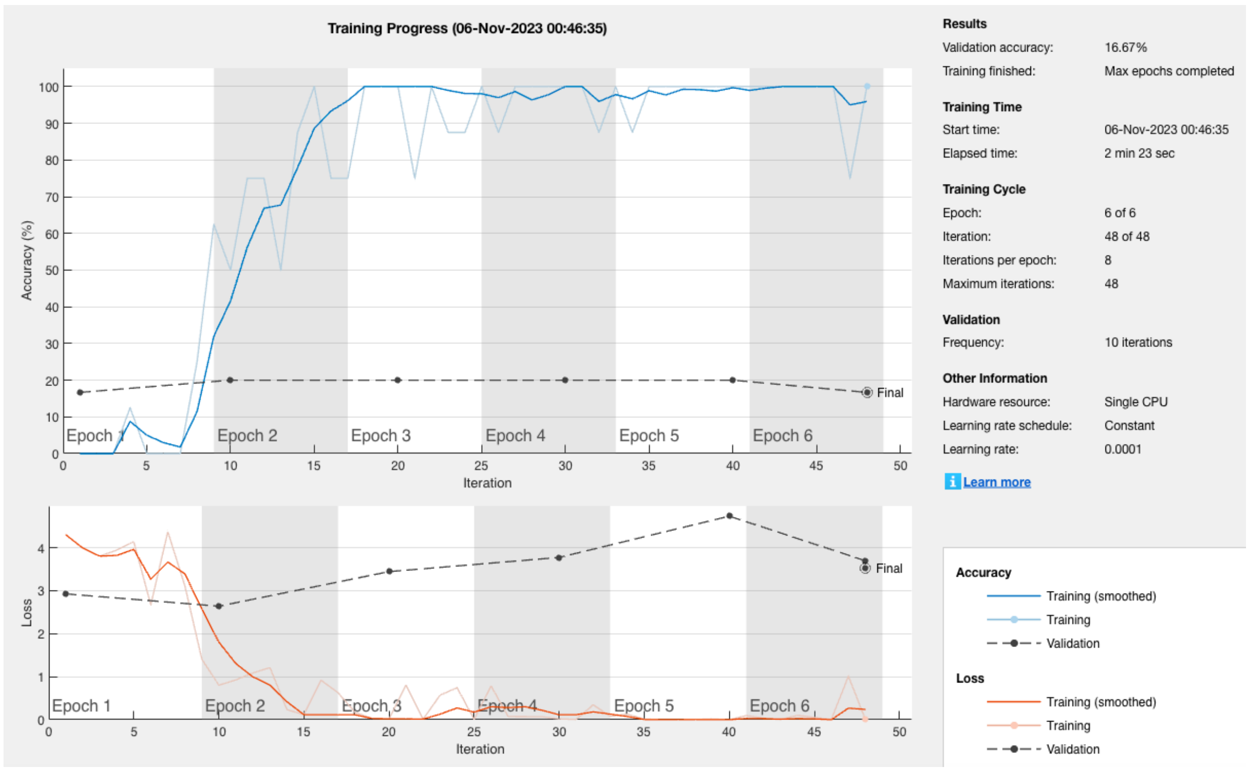
Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Figure 14: Accuracy rates for test run of 1000 images with increased feature size in convolution layers

While this accuracy rate can still be seen as low, the percentage of accuracy shows a large improvement from earlier versions. This improvement details some important information about the correspondence between the number of layers and features in each layer with the overall accuracy.

Another important discovery to note is how a matching layering design can affect the overall accuracy on systems using different training data sizes. After finding the layering configuration providing the highest accuracy rate for a training dataset of 1000 images, I repeated the process of training the system with a dataset of 100 total images and a dataset of 10,000 total images. The following graphs display the results after training the system with a dataset of 100 total images.



Results

Validation accuracy:	16.67%
Training finished:	Max epochs completed

Training Time

Start time:	06-Nov-2023 00:46:35
Elapsed time:	2 min 23 sec

Training Cycle

Epoch:	6 of 6
Iteration:	48 of 48
Iterations per epoch:	8
Maximum iterations:	48

Validation

Frequency:	10 iterations

Other Information

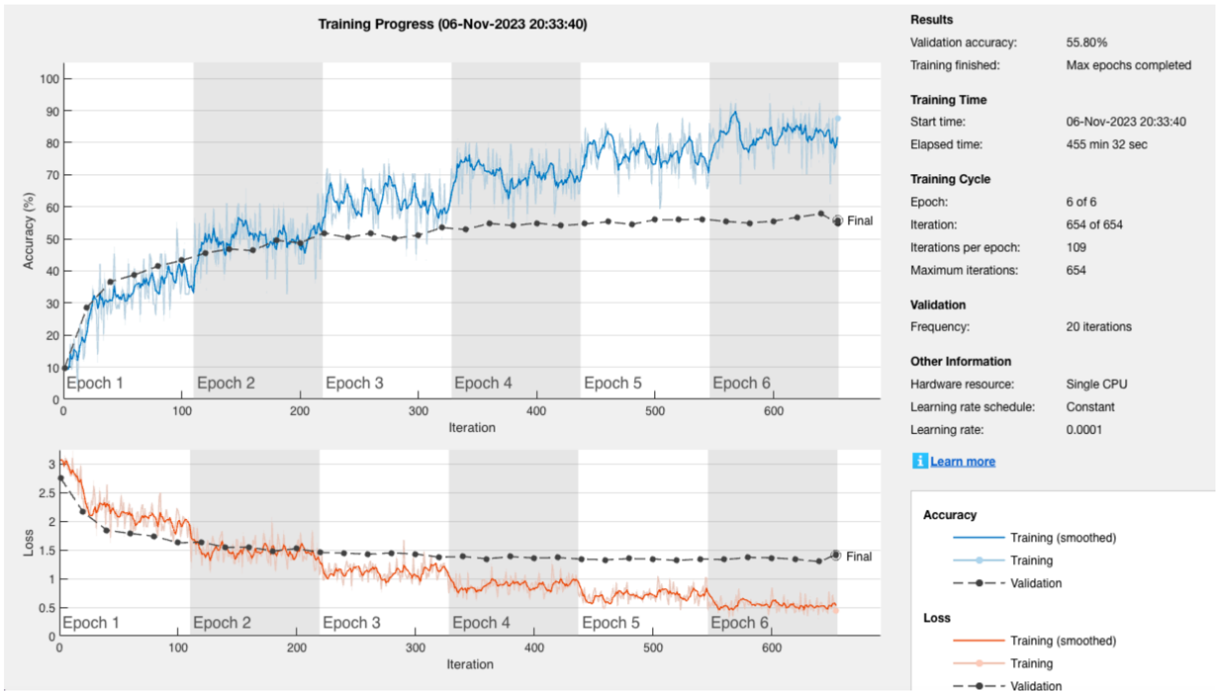
Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Accuracy
- Training (smoothed)
- Training
- Validation

Loss
- Training (smoothed)
- Training
- Validation

Figure 15: Accuracy rates for test run of 100 images with highest performance layering configuration

The smaller dataset performed poorly in comparison to the larger image set. The final training accuracy was almost 100%, while the final validation accuracy calculated to only 16.67%. The overabundance of layers in comparison to the given training data created an imbalance in the system and produced a large amount of overfitting.



Results

Validation accuracy:	55.80%
Training finished:	Max epochs completed

Training Time

Start time:	06-Nov-2023 20:33:40
Elapsed time:	455 min 32 sec

Training Cycle

Epoch:	6 of 6
Iteration:	654 of 654
Iterations per epoch:	109
Maximum iterations:	654

Validation

Frequency:	20 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Figure 16: Accuracy rates for test run of 10,000 images with highest performance layering configuration

As seen in the graph above, when utilizing a training dataset of 10,000 images the system resulted in a final validation accuracy of 55.8%. While this accuracy rate still has room for improvement, the test run is unique by the limited level of overfitting. Unlike the more constraining datasets, the training dataset of 10,000 images provided an improved balance with the layering configuration, resulting in a very limited amount of overfitting.

These test runs demonstrate how accuracy not only depends on how the layers in a CNN are configured, yet also on the number of images used to train the system. When categorizing images containing an abundance of features and requiring multiple convolution and hidden layers, it is best to train the network using large data sets. Training the system with a limited dataset leads to a greater potential for overfitting and a smaller validation accuracy. As seen in the previous test runs, the largest dataset of 10,000 images provided the highest performance on the low-end system.

ADDITIONAL COMPLICATIONS AND POTENTIAL FUTURE WORKS

As previously mentioned, the current state of the program suffers from issues of overfitting. Additional time would be required to find better hyperparameters necessary to increase accuracy rates. Additional changes could be made to improve accuracy in future works. For example, this experiment could be replicated and conducted on a high-end system. Obtaining a more powerful machine with increased resources, such as larger memory storage, could allow for a larger training data set and increased computational resources. Furthermore, future additions or similar projects could focus on utilizing different programming languages. While MATLAB is a powerful programming language and has advantages when creating a CNN, other languages such as Python have also proven useful in the field of computer vision.

CONCLUSION

In conclusion, as computer vision becomes more prevalent in today’s everyday usage, the public is relying on accurate results from object recognition algorithms. However, there are multiple attributes that accumulate to influence the accuracy levels of a computer vision system’s results.

The amount and type of neural network layers should develop the system’s prediction calculations without becoming overly extensive regarding the amount of training data. As seen in the experiment, an overabundance of layers could lengthen the system’s time in extraneous and unnecessary calculations as well as create an issue of overfitting, harming the overall accuracy of the system. A lacking number of layers could also negate the system’s accuracy by not providing enough calculations or thorough determinations of details within an image.

The computer vision system should not be expected to provide perfect predictions on the first run-through. Therefore, the system should be equipped with optimization methods. The data is better understood and patterns more easily recognizable by using backpropagation, allowing the nodes to be adjusted using gradient descent to find the minimum cost for each weight.

The training data should be complete, providing enough examples and details so that the system is not confused or shocked by any objects present in the image. The training data set should not be overly numerous or lacking in comparison to the number of layers. However, the users of the system should also be aware that as time goes on and the world develops, the system should be able to recognize new inventions or additions in the data. The system should be somewhat kept up with for the inclusion of new data or ability to create new categories to keep the system from becoming outdated.

Modern systems are developing at a rapid pace towards providing an increased accuracy in object classifications. Presently, people are attempting to incorporate object recognition systems into scenarios from every aspect of life for the ease and accessibility of quick, accurate results. However, programmers and other developers still have a long way to go in increasing the accuracy of these systems; a job that will likely take years to accomplish since not even the human visual system is 100% perfect.

REFERENCES

[1] Alessio, Corrado (n.d.) *Animals - 10.* Kaggle. Retrieved Sept. 20, 2023, from https://www.dropbox.com/s/j5xdb2hmkp900gz/corrado-tesina-en.pdf?dl=0.

[2] Ali, Muhammad Shoaib. “Flattening CNN Layers for Neural Network and Basic Concepts.”, *Medium*, Medium, 23 June 2022, Retrieved Sept. 20, 2023, from medium.com/@muhammadshoaibali/flattening-cnn-layers-for-neural-network-694a232eda6a.

[3] Bandyopadhyay, Hmrishav. (2022, June 9). “What is Computer Vision? [Basic Tasks & Techniques]”, *V7*. Retrieved Sept. 20, 2023, from https://www.v7labs.com/blog/what-is-computer-vision.

[4] “Computer Vision Market Size, Share & Growth [2023 Report].” *Computer Vision Market Size, Share & Trends Analysis Report By Component, By Product Type, By Application, By Vertical (Automotive, Healthcare, Retail), By Region, And Segment Forecasts, 2023-2030*, Grand View Research. Retrieved Sept. 20, 2023, from www.grandviewresearch.com/industry-analysis/computer-vision-market.

[5] Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai and Fei-Fei, Li, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

[6] “Gradient Descent: An Optimization Technique in Machine Learning.” *Hire the World’s Most Deeply Vetted Remote Developers*, Turing Enterprises Inc, 11 Apr.

2022. Retrieved Sept. 20, 2023, from www.turing.com/kb/gradient-descent-optimization-technique-in-machine-learning.

[7] Huang, T. S. (n.d.). “Computer vision: Evolution and Promise – CERN”. Retrieved Sept. 20, 2023, from https://cds.cern.ch/record/400313/files/p21.pdf.

[8] Khosla, Aditya, Jayadevaprakash, Nityananda, Yao, Bangpeng and Fei-Fei, Li. Novel dataset for Fine-Grained Image Categorization. First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2011.

[9] Kwiatkowski, Robert. “Gradient Descent Algorithm - A Deep Dive.” *Medium*, Towards Data Science, 13 July 2022. Retrieved Sept. 20, 2023, from towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21.

[10] “Multi-Class Neural Networks: Softmax | Machine Learning | Google for Developers.” *Google*, Google. Retrieved Sept. 20, 2023 from developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax#:~:text=Softmax%20is%20implemented%20through%20a,layer%20within%20a%20neural%20network.

[11] Otmseo. “2022 Self-Driving Car Accident Statistics - 1-800-INJURED.” *Self Driving Car Accident Statistics*, 2 June 2022. Retrieved Sept. 20, 2023, from 1800injured.care/self-driving-car-accident-statistics/.

[12] "Output Size.", *Fully Connected Layer – MATLAB*. Retrieved Sept. 20, 2023, from www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html.

[13] Pasricha, Sudeep, and Balasubramaniam, Abhishek. "Object Detection in Autonomous Vehicles: Status and Open Challenges", Colorado State University, 19 Jan, 2022. Retrieved Sept. 20, 2023, from arxiv.org/abs/2201.07706.

[14] Pinto, Nicolas, Cox, David, and DiCarlo, James. "Why Is Real-World Visual Object Recognition Hard?", *PLoS Computational Biology.* Edited by Karl J Friston, U.S. National Library of Medicine, Jan. 2008. Retrieved Sept. 20, 2023, from www.ncbi.nlm.nih.gov/pmc/articles/PMC2211529/.

[15] Ronaghan, Stacey. "Deep Learning: Which Loss and Activation Functions Should I Use?" *Medium*, Towards Data Science, 1 Aug. 2019. Retrieved Sept. 20, 2023, from towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8.

[16] Says, Saurabh. "What Is Backpropagation?: Training a Neural Network." *Edureka*, 15 Nov. 2022. Retrieved Sept. 20, 2023, from www.edureka.co/blog/backpropagation/.

[17] Srinivasan, Aishwarya V. "Stochastic Gradient Descent-Clearly Explained !!" *Medium*, Towards Data Science, 7 Sept. 2019. Retrieved Sept. 20, 2023, from towardsdatascience.com/stochastic-gradient-descent-clearly-explained-

53d239905d31#:~:text=%E2%80%9CGradient%20descent%20is%20an%20iterative,of%20the%20function%20to%200.

[18] Titus, Tienaah, Adu-Gyamfi, Okyere, Yaw, Asare, Kwasi, Sampson, & Sharma, Anuj. 2017. “Automated Vehicle Recognition with Deep Convolutional Neural Networks”. *Transportation Research Record: Journal of the Transportation Research Board* (2645). Pp. 113-122. Retrieved Sept. 20, 2023, from https://journals.sagepub.com/doi/pdf/10.3141/2645-13.

[19] “Train Network.” *MATLAB & Simulink Example*, Retrieved Sept. 20, 2023, from www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html.

[20] Weiwei Zhang, Jian Sun, and Xiaoou Tang. 2008. “Cat Head Detection - How to Effectively Exploit Shape and Texture Features”, *Proc. of European Conf. Computer Vision*, vol. 4. Pp. 802-816.

[21] “What Is Computer Vision?” *IBM*, Retrieved Sept. 24, 2023, from www.ibm.com/topics/computer-vision.

[22] Zhang, Jinglan, Alzubaidi, Laith, Humaidi, Amjad, Al-Dujaili, Ayad, Duan, Ye, Al-Shamma, Omran, Santamaria, J., Fadhel, Mohammed, Al-Amidie, Muthana, & Farhan, Laith. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions”, *Journal of Big Data*. SpringerLink. March 31, 2021. Retrieved Sept. 20, 2023, from https://link.springer.com/article/10.1186/s40537-021-00444-8.

APPENDIX A: CODE

Code A: MATLAB program to calculate Gradient Descent

```
func = @(x)x^2-6*x+1;

derivativeFunction = @(x)2*x - 6;

minAcceptance = .01;

limit = 100;

%learning rate

N = .1;

allX = [];

allY = [];

cnt = 0;


%start at a random point in the graph

x = -3;


%find first derivative of the function (function for slope)

gradient = derivativeFunction(x);


%find gradient descent using derivative of the function

while(abs(gradient) > minAcceptance && cnt < limit)

allX(end + 1) = x;

allY(end + 1) = func(x);
```

```
cnt = cnt + 1;


%go to next point on graph using learning rate
x = x - N * gradient;
gradient = derivativeFunction(x);
end


%plot the function and each determination point visited in
the function
fplot(func, [-6,8])
title("Solving for Gradient Descent");
xlabel("XRange");
ylabel("YRange");
hold on
plot(allX, allY, "r*-")
hold off
```

Code B: Computer Vision Object Recognition program

```
%{
This is a program to create a computer vision object
recognition system. The goal is to read in an existing
image data set to train the neural network and determine
```

```
which hyperparameters give the most desirable accuracy
results.

The hyperparameters to change include changing max vs
average pooling layer, and changing the number of hidden
layers versus the quantity of training data
%}

%getting the training data image set
totalImageSet =
imageDatastore('CET_CV_ImageSet/100_DataSet','IncludeSubfol
ders',true,'LabelSource','foldernames');
%will show how many times a label is used in the image
dataset
%countEachLabel(totalImageSet);

%resize all training data to be the same size
imSize = 300;
%{
while hasdata(totalImageSet)
    [currentImage, info] = read(totalImageSet);
    currentImage = imresize(currentImage, [imSize,
imSize]);
    imwrite(currentImage, info.Filename);
```

```
end
reset(totalImageSet)
%}

[trainingImages, validationImages] =
splitEachLabel(totalImageSet, 70, 'randomize');

%setting up the initial layers
layers = [
    %corresponds to the size of the image
    imageInputLayer([imSize imSize 3])

    %first convolution layer
    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer()
    %first pooling layer
    maxPooling2dLayer(2, 'Stride', 2)

    %second convolution layer
    convolution2dLayer(3, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer()
    maxPooling2dLayer(2, 'Stride', 2)

    %third convolution layer
    convolution2dLayer(3, 128, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer()

    maxPooling2dLayer(2, 'Stride', 2)

    %fourth convolution layer
    convolution2dLayer(3, 256, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer()

    maxPooling2dLayer(2, 'Stride', 2)

    %fifth convolution layer
    convolution2dLayer(2, 512, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer()

    %transition to fully connected layers
    %fully-connected hidden layers correspond with number
of final classification options
```

```
    fullyConnectedLayer(256)

    fullyConnectedLayer(512)

    %dropoutLayer(.2)

    fullyConnectedLayer(512)

    fullyConnectedLayer(512)

    %dropoutLayer(.2)

    fullyConnectedLayer(256)

    fullyConnectedLayer(10)


    %training data should only result in one classification

    softmaxLayer

    %output layer

    classificationLayer

];


%extra hyperparameter options for training data

%using stochastic gradient descent (sgdm)

options = trainingOptions('sgdm', ...

     'InitialLearnRate', 0.0001, ...

    'MaxEpochs',6,'Shuffle','every-epoch',...

    'MiniBatchSize',32,...

    'ValidationData',validationImages, ...

    'ValidationFrequency', 10, 'Verbose', true,...

    'Plots','training-progress');
```

```
%use the layers and other hyperparameters to train the cnn
using
%the image dataset of labeled images
cnn = trainNetwork(trainingImages, layers, options);

%use the image dataset of validation labeled images to
determine
%accuracy of the cnn's object recognition algorithm
prediction = classify(cnn,validationImages);
validationLabels = validationImages.Labels;

percentOfAccuracy = sum(prediction ==
validationLabels)/numel(validationLabels);
disp("The percentage of accuracy is "+ percentOfAccuracy)
```

Code C: Program to read in an image

```
img = imread("Western_Cup.png");
%img = imresize(img, .04)
imshow(img);
```