June 2016

An Exercise and Sports Equipment Recognition System

Siddarth Kalra  
The University of Western Ontario

Supervisor  
Charles X. Ling  
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

© Siddarth Kalra 2016

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation


This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact tadam@uwo.ca, wlswadmin@uwo.ca.
Abstract

Most mobile health management applications today require manual input or use sensors like the accelerometer or GPS to record user data. The onboard camera remains underused. We propose an Exercise and Sports Equipment Recognition System (ESRS) that can recognize physical activity equipment from raw image data. This system can be integrated with mobile phones to allow the camera to become a primary input device for recording physical activity. We employ a deep convolutional neural network to train models capable of recognizing 14 different equipment categories. Furthermore, we propose a preprocessing scheme that uses color normalization and denoising techniques to improve recognition accuracy. Our best model is able to achieve a top-3 accuracy of 83.3% on the test dataset. We demonstrate that our model improves upon GoogLeNet for this dataset, the state-of-the-art network which won the ILSVRC 2014 challenge. Our work is extendable as improving the quality and size of the training dataset can further boost predictive accuracy.

Keywords: Object recognition, machine learning, convolutional neural networks, color normalization
I would like to acknowledge my supervisor, Dr. Charles X. Ling, for his feedback and guidance during this thesis. He has been a tremendous help throughout this process and I wouldn’t have been able to complete this thesis without his experience and support.

I would like to thank Chunfang Wang for her help with the Caffe deep learning framework. Her continued help throughout this process was invaluable in increasing my understanding of the framework. Also, thank you to Yunming Xia and Maggie Feng for helping in collecting images for the test dataset. Yunming was also generous enough to help manually set the ground truth labels for many of the samples in the training dataset.

Lastly, I want to thank my parents, Neena and Suresh Kalra, and my sister, Shreya Kalra, for their continued love and support.
## Contents

Abstract ii
Acknowlegements iii
List of Figures vi
List of Tables x
List of Appendices xii
List of Abbreviations xiii

1 Introduction 1
   1.1 GlucoGuide - A Mobile Diabetes Management System ...................... 2
   1.2 Previous Work ................................................................. 3
   1.3 Dataset .......................................................... 4
   1.4 Challenges ................................................................. 5
   1.5 Proposed Solutions ...................................................... 10
   1.6 Contributions ............................................................ 12
   1.7 Thesis Structure ........................................................... 13

2 Review of Artificial Neural Networks and Deep Learning for Object Recognition 14
   2.1 Artificial Neural Networks .................................................. 14
       2.1.1 Perceptrons .......................................................... 15
       2.1.2 Sigmoid Neurons ...................................................... 16
       2.1.3 Multilayer Perceptrons .............................................. 16
       2.1.4 Backpropagation and Gradient Descent ............................ 17
   2.2 Convolutional Neural Networks ........................................... 22
       2.2.1 2D Convolution .......................................................... 23
       2.2.2 Input Layer ............................................................. 23
       2.2.3 Convolutional Layer .................................................. 24
       2.2.4 Subsampling Layer ................................................... 25
       2.2.5 Activation Layer ...................................................... 26
       2.2.6 Fully Connected Layer .............................................. 26
       2.2.7 Loss Layer ............................................................... 27
       2.2.8 Traditional Architectures .......................................... 28
   2.3 Recent Advances in Deep Learning for Object Recognition ............. 28
3 Training Deep Networks for Exercise and Sports Equipment Recognition

3.1 Training a Baseline Model

3.2 Hyperparameter Tuning

3.3 Augmenting the Baseline Architecture

3.3.1 The Intuition and Theory behind GoogLeNet

3.3.2 Architecture Augmentation Results

3.4 Regularization to Reduce Overfitting

3.4.1 Review of L2 Regularization

3.4.2 Review of Dropout

3.4.3 Results from Applying Regularization

3.5 Transfer Learning

3.5.1 Results from Finetuning

3.6 Summary of Validation Results

4 Review of Color Normalization and Denoising to Improve Object Recognition

4.1 Histogram Equalization

4.1.1 Adaptive Histogram Equalization

4.1.2 Contrast Limited Adaptive Histogram Equalization (CLAHE)

4.2 Non-local Means Denoising

5 Performance on Test Data

5.1 Best Performing Models Without Finetuning

5.2 Best Performing Models With Finetuning

5.3 Results from Applying Color Normalization and Denoising

5.3.1 Results from Applying CLAHE

5.3.2 Results from Applying CLAHE and NL-means Denoising

5.4 Addressing Challenges

6 Conclusions and Discussion

6.1 Conclusions

6.2 Future Work

Bibliography

A More Recognition Results on Test Images from Model I

Curriculum Vitae
List of Figures

1.1 The GlucoGuide app (iOS App Store) provides three input methods to record dietary information. .......................... 2
1.2 A mockup of how the GlucoGuide app would function with integration of exercise and sports recognition. .......................... 3
1.3 The MyFitnessPal app (iOS App Store) is the market leader in the domain of health management. It currently relies on manual user input to track physical activity. .......................... 4
1.4 The 14 different categories that ESRS is capable of recognizing. .......................... 6
1.5 3 of the 14 categories are used in racquet sports and thus are very similar to each other in physical appearance. This similarity may cause confusion, especially when viewed from a side-on angle, for a classifier. .......................... 7
1.6 There is a considerable amount of visual variance that can exist within individual categories. We provide three examples each found within the dataset from the Spinning Bike (top row) and Golf Club Head (bottom row) categories to demonstrate the type of variance that exists. This intra-category variation poses a challenge as a classifier must be capable of recognizing the correct category despite the existence of subcategories within the dataset. .......................... 8
1.7 There is a disparity between the training and validation datasets in comparison to the test dataset. Many of the images within the former datasets contain annotations and animations. We have provided four examples from the dataset. This may mislead the classifier and teach it features that do not exist in the test dataset. .......................... 8
1.8 The images found within the training and validation datasets for the Kayak category are very different than the ones found within the test dataset. The former datasets contain many examples of kayaks found within bodies of water (top row) while the latter dataset does not contain any depictions of bodies of water (bottom row). This disparity may make it difficult for a classifier to correctly identify kayaks within the test dataset. .......................... 9
1.9 The dataset contains many examples where a single image contains multiple categories. This can cause a network to learn misleading features that are not native to the ground truth category. To be consistent, we made sure to label the foreground category as the ground truth. Thus, an ideal classifier needs to have the ability to focus on the category that is the foreground and ignore the one in the background. .......................... 10
1.10 All the categories found within this dataset have been designed for human use. Thus, naturally we find that many images within the dataset show humans manipulating an object (top row). Furthermore, natural images tend to contain a good amount of background clutter (bottom row) that adds noise to the input data. An ideal classifier needs to have the ability to disregard the human figures and background clutter.

1.11 Smartphones tend to produce a relatively large amount of low contrast images as the devices are regularly used by amateurs in varying light conditions. Our test dataset contains many examples of such images. Such images can be difficult to classify as objects can easily be masked or hidden.

2.1 A simple artificial neural network that is fully connected. Every neural network has one input layer, one output layer and at least 1 hidden layer. It is called a fully connected network because each neuron in layer $l$ connects to every neuron in layer $l + 1$.

2.2 A perceptron that takes $x_m$ inputs along with $w_m$ weights to produce output($v$). Larger weights have more influence on the output.

2.3 A comparison of activation functions. The perceptron activation function can only produce an output of 0 or 1 while the sigmoid activation is smoother as it allows any value between 0 and 1. This smoothness is a desirable trait for a neuron as small changes in input weights lead to small changes in the output. In contrast, small weight changes to the input of a perceptron function can lead to large changes in the output.

2.4 An example of a 4 layer Multilayer perceptron with 2 hidden layers. The input layer has $n$ neurons and the output layer has $n'$ neurons.

2.5 Neural network learning has two stages of processing. The forward pass determines the network’s output. The backward pass first calculates the overall error and then uses backpropagation and gradient descent to update the weights to minimize this error. These two stages are repeated until the error is minimized and the best possible performance is achieved.

2.6 The surface area of a loss function with a local minima and a global minima. Backpropagation identifies the gradient while gradient descent traverses the surface area to locate the global minima.

2.7 The notation $w_{ij}^l$ and $p_i^l$ is used to identify weights and outputs respectively. Such a weight connects the $i^{th}$ neuron in the $l^{th}$ layer to the $j^{th}$ in the $(l - 1)^{th}$ layer.

2.8 Edge Detection by convolving a filter on an input image. The makeup of the filter impacts the result of the convolution. In this case, the filter reveals all the edges within the image.

2.9 In a fully connected layer, each neuron connects to all neurons in the previous layer. When the input datum is an image, this means that each neuron is exposed to the entire image. Alternatively, in a locally connected layer, a local receptive field is formed as each neuron only connects to a certain subset of neurons from the previous layer. This allows such layers to learn local traits of an image.
2.10 A visualization of an image from the BaseballGlove category being processed by a modified version of GoogLeNet (Model I). (a) The input image supplied to the CNN (b) Visualization of the filters produced by the first convolutional layer (c) Visualization of the output produced by the first convolutional layer.

2.11 A standard convolutional neural network (CNN) contains groups of convolution and subsampling layers with a fully connected layer at the end to perform high level reasoning.

2.12 Two common pooling operations used in subsampling layers are depicted. We show the result of sliding a $2 \times 2$ filter across a $4 \times 4$ feature map. Max pooling takes the maximum value found within the $2 \times 2$ patch while average pooling takes the average of the values within the patch.

3.1 Results for Model A (baseline). The large gap between the train and validation loss indicates that overfitting is occurring.

3.2 Results for Model B. The large gap between the train and validation loss indicates that overfitting is occurring.

3.3 The mlpconv Layer, proposed by [35], employs multiple fully connected layers to convolve over the input data. It can be interpreted as an extra $1 \times 1$ convolutional layer with ReLU activation. Its benefits include reducing the number of network parameters and an improved ability to handle nonlinear data.

3.4 Inception module structures (adapted from [59])

3.5 The full GoogLeNet architecture (adapted from [59])

3.6 Results for Model C. The large gap between the train and validation loss indicates that overfitting is occurring.

3.7 Results for Model D. The large gap between the train and validation loss indicates that overfitting is occurring.

3.8 Dropout, a regularization procedure, works by deactivating a random subset of neurons within a network. In this depiction, three neurons have been disabled. This means that each input datum or each mini-batch is exposed to a slightly different network structure making it difficult for neurons to co-adapt.

3.9 Results for Model E. The large gap between the train and validation loss indicates that overfitting is occurring.

3.10 Results for Model F. The gap between the train and validation loss is significantly smaller relative to all other models indicating that overfitting has been curtailed.

3.11 Results for Model G. The large gap between the train and validation loss indicates that overfitting is occurring.

3.12 Results for Model H. The large gap between the train and validation loss indicates that overfitting is occurring.

3.13 Results for Model I. The large gap between the train and validation loss indicates that overfitting is occurring.

3.14 The top-1 and top-5 accuracies achieved by each model on the validation dataset.
4.1 The three different workflows we used for object recognition. The original workflow involves directly submitting the input image to the classification model for prediction. The other two workflows involve CLAHE and NL-means pre-processing.

4.2 Histogram equalization transforms the pixel intensity histogram of an image so that the pixel intensities are more uniformly distributed by applying a cumulative distribution function (CDF) to the original histogram.

4.3 Adaptive Histogram Equalization functions by applying the histogram equalization technique on $n \times n$ patches throughout the image.

4.4 This depiction shows how CLAHE can increase contrast and NL-means denoising can help reduce noise after CLAHE has been applied.

5.1 Model G was the best performing non-finetuned model for the test dataset. Here, it is compared with the non-finetuned baseline model.

5.2 Model I was the best performing finetuned model for the test dataset. Here, it is compared with the finetuned baseline model.

5.3 This chart depicts the impact that CLAHE and NL-means denoising had on the top-1, top-3 and top-5 accuracies of the best non-finetuned ESRS model (Model G) and the non-finetuned baseline (Model A).

5.4 This chart depicts the impact that CLAHE and NL-means denoising had on the top-1, top-3 and top-5 accuracies of the best finetuned ESRS model (Model I) and the finetuned baseline (Model H).

5.5 Low contrast images from the test dataset that were recognized correctly due to CLAHE preprocessing. For each image set, the left image is the original and the right image is the result of applying CLAHE. The correct label is printed under each image set and the assigned probability for the correct label is highlighted by the red bar, if it is present within the top-5 results.

5.6 Various images from the test dataset that contained significant noise (e.g. humans, background clutter, multiple categories). The correct label is printed under each image and the assigned probability for the correct label is highlighted by the red bar, if it is present within the top-5 results. See Appendix A for more examples.

A.1 Recognition results on test images. The correct label is printed under each image and the assigned probability for the correct label is highlighted by the red bar, if it is present within the top-5 results.
List of Tables

1.1 A numerical breakdown of all the categories and their respective train, validation and test sample sizes. ................................................................. 5

3.1 Initial hyperparameter selections for the baseline model (Model A). ............. 31
3.2 Results for Model A (baseline) garnered by training the standard GoogLeNet architecture. These results are used as a benchmark to measure the success of all other models. Note: The Loss Difference column indicates the difference between the train and validation loss. ................................................................. 31
3.3 Results for Model B. The same architecture as Model A is used but the learning rate hyperparameter is reduced from $1.0 \times 10^{-2}$ to $1.0 \times 10^{-4}$. .................. 32
3.4 Results for Model C. The base GoogLeNet architecture is augmented by adding another Inception module at the end of architecture but before the final fully connected layer. ................................................................. 35
3.5 Results for Model D. The base GoogLeNet architecture is augmented by adding another fully connected layer at the very end of the network. ................. 37
3.6 Results for Model E. We increase the regularization used by GoogLeNet by increasing the dropout ratio in the last fully connected layer from 40% to 55%. 40
3.7 Results for Model F. We increase the regularization used by GoogLeNet by increasing the dropout ratio in the last fully connected layer from 40% to 70% and increasing the weight decay (L2 regularization) hyperparameter from $2.0 \times 10^{-4}$ to $2.0 \times 10^{-3}$. ................................................................. 40
3.8 Results for Model G. We increase the regularization used by GoogLeNet by increasing the dropout ratio in the last fully connected layer from 40% to 70% . 41
3.9 To setup a benchmark, the baseline model (Model A) was finetuned. Table 3.10 and Figure 3.12 present the results for this finetuned model, called Model H. Our best performing model (Model G) was also finetuned so that it could be compared with the finetuned baseline. Table 3.11 and Figure 3.13 present the results for this model, called Model I. ................................. 43
3.10 Results for Model H. This is the finetuned version of the baseline model (Model A). ................................................................. 43
3.11 Results for Model I. This is the finetuned version of Model G. .................. 45
3.12 A summary of results for the training and validation datasets. Note: The Loss Difference column indicates the difference between the train and validation loss. 46
5.1 Model G was the best performing non-finetuned model for the test dataset. Here, it is compared with the non-finetuned baseline model. We also show the results after excluding the Kayak and Golf Club Head categories to highlight that both models were poor at recognizing these two categories as the overall accuracy rates improve significantly if they are excluded. ........................................ 53

5.2 Confusion Matrix for Model A, the non-finetuned baseline. .......................... 54

5.3 Confusion Matrix for Model G, the best ESRS model (non-finetuned). ........ 54

5.4 Model I was the best performing finetuned model for the test dataset. Here, it is compared with the finetuned baseline model. We also show the results after excluding the Kayak and Golf Club Head categories to highlight that both models were poor at recognizing these two categories as the overall accuracy rates improve significantly if they are excluded. ........................................ 55

5.5 Confusion Matrix for Model H, the finetuned baseline ................................. 56

5.6 Confusion Matrix for Model I, the best ESRS model (finetuned). ............... 56

5.7 The top-1, top-3 and top-5 accuracies achieved by the best ESRS models on the test dataset after preprocessing with CLAHE and NL-means denoising. . . . 57
List of Appendices

Appendix A More Recognition Results on Test Images from Model I . . . . . . . . . . . 70
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESRS</td>
<td>Exercise and Sports Equipment Recognition System</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial neural network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional neural network</td>
</tr>
<tr>
<td>ILSVRC</td>
<td>ImageNet Large Scale Visual Recognition Challenge</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer perceptron</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified linear units</td>
</tr>
<tr>
<td>FC layer</td>
<td>Fully connected layer</td>
</tr>
<tr>
<td>NIN</td>
<td>Network in Network</td>
</tr>
<tr>
<td>GLM</td>
<td>Generalized linear model</td>
</tr>
<tr>
<td>Loss Difference</td>
<td>Percentage difference between the train and validation loss</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative distribution function</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, green and blue</td>
</tr>
<tr>
<td>LHS</td>
<td>Luminance, hue and saturation</td>
</tr>
<tr>
<td>HSI</td>
<td>Hue, saturation and lightness</td>
</tr>
<tr>
<td>AHE</td>
<td>Adaptive Histogram Equalization</td>
</tr>
<tr>
<td>CLAHE</td>
<td>Contrast Limited Adaptive Histogram Equalization</td>
</tr>
<tr>
<td>NL-means Denoising</td>
<td>Non-local Means denoising</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this thesis, we describe in detail the theory and methodology behind the Exercise and Sports Equipment Recognition System (ESRS), which is capable of recognizing sports and exercise equipment within natural images and can easily be integrated with mobile devices. Many users still find today’s mobile health management systems tedious to use for day-to-day tracking purposes. Very little research and development work, both in academia and the commercial sector, has been done in using the smartphone camera as the main input device for tracking health related information. This is why we propose ESRS as the main input system for an alternate logging framework that can track information about physical activity by using a smartphone camera. Our work takes inspiration from, and will eventually integrate with, GlucoGuide [37, 20] - a mobile diabetes management system that already allows users to track dietary information by taking pictures of food items.

ESRS has the capability to recognize 14 different categories (see Figure 1.4) that patients might commonly use during a workout. They include different activities that patients might perform to stay physically active. The two types of exercise regimes, resistance and aerobic training, are both important for patients [15, 16, 9, 57, 38]. This is why equipment from both regimes is included. For example, dumbbells and barbells are used for resistance training, while swimming pools and running shoes are used for aerobic training.

In order to create a generalized classifier, capable of recognizing the 14 categories within raw images, we propose a learning based approach by employing a deep convolutional neural network (CNN) to create a predictive model. Recognizing objects within natural images is an easy task for human beings but not for a computer program that takes an algorithmic approach. Thus, we turn to machine learning to solve the problem. Specifically, we refine and improve upon the winning architecture of the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), GoogLeNet [59]. A wide variety of challenges need to be overcome to produce a viable model. They include various discrepancies between the training and test samples, overfitting due to the relatively small dataset, small inter-category and large intra-category variations, and the low contrast images commonly produced by smartphones due to varying light conditions. Nonetheless, we show that our proposed solution performs well for this dataset by improving on the state-of-the-art, reducing overfitting via regularization and demonstrating that prediction accuracy can be boosted by applying color normalization and denoising techniques.

In this chapter, Section 1.1 describes GlucoGuide; Section 1.2 introduces relevant previ-
Figure 1.1: The GlucoGuide app (iOS App Store) provides three input methods to record dietary information.

ous work; Section 1.3 presents the dataset constructed for this work; Section 1.4 outlines the challenges that need to be addressed in order to create a generalized model capable of making accurate predictions; Section 1.5 proposes solutions to tackle the challenges; Section 1.6 highlights our major contributions; and in Section 1.7 we outline how the rest of the chapters within this paper are structured.

1.1 GlucoGuide - A Mobile Diabetes Management System

GlucoGuide [37, 20] is a mobile diabetes management system that analyzes diet and physical activity data to provide non-medical advice to users with the goal of improving the health of diabetic patients. The application is available for download via official app stores for smartphones capable of running Android or iOS. In order to keep track of their progress, diabetic patients must keep a daily log of their exercise and diet choices. As is the case with many applications that require user input, most of the data has to be logged manually by the user. However, this can be a tedious task given the relatively small form factor of mobile devices. This is why the diet tracking section of the app provides three different forms of input: search via the digital keyboard, barcode scanner or food recognition via the smartphone’s camera (see Figure 1.1). The first input method is similar to how many of the other health-oriented apps provide user input. That is, a user can search through a database accessible via the phone to locate and log a food item. The barcode method is also widely implemented by many apps and allows users to scan the barcode of items purchased from grocery stores. There are limitations with these two methods as many food items don’t have barcodes (e.g., food bought at restaurants and cafes) or the food item may not be part of the back-end database. The camera method works quite differently as it allows the user to take a picture of their food. Then the app processes the raw image data through a predictive model that attempts to recognize the food item(s) in the image. If the recognition is successful, the nutritional data for the food item(s) is provided to the user. In the age of social media, many users already regularly share pictures of their food with family and friends. Thus, this input method is not just simpler; it can also be enjoyable and fascinating for the user.

The success of the food recognition method is the inspiration behind our Exercise and Sports Equipment Recognition System (ESRS). GlucoGuide also needs to log physical activity information, which is done manually today. We came to the conclusion that the smartphone
1.2 Previous Work

Today’s smartphones house a variety of sensors such as an accelerometer, gyroscope, GPS, microphone and camera. Many applications have been developed that use these tools to gather user information [30]. In terms of the health and fitness industry, applications have also been developed that use a smartphone’s sensors to gather physical activity information from the user [11, 36]. There are also many examples of object recognition being employed in a variety of different fields. However, to our knowledge, there is no commercial application available today that directly examines raw image data to track a user’s exercise progress. MyFitnessPal [42], the market leader in the domain of health management, currently relies on manual input from the user to track physical activity (see Figure 1.3). On the academic side, there are vast quantities of papers that focus on general object recognition [62, 34, 17]. In the last few years, landmark research has been conducted by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton [29] in this field. We cover their work in greater deal in Section 2.3. Most recently, GoogLeNet [59], winner of ILSVRC 2014, has set the state-of-the-art in the field. We cover the theory behind GoogLeNet in Section 3.3.1.

However, very little work has been conducted to integrate calorie and exercise tracking systems on smartphones with object recognition systems that possess the capability to recog-
Figure 1.3: The MyFitnessPal app (iOS App Store) is the market leader in the domain of health management. It currently relies on manual user input to track physical activity.

Our proposed Exercise and Sports Equipment Recognition System (ESRS) is the first step on the road to such an integrated ecosystem.

1.3 Dataset

This object recognition task is complex and challenging (see Section 1.4). Thus, we use the most powerful method in object recognition, machine learning, to create a predictive model capable of recognizing exercise and sports equipment within raw images. To train predictive models capable of recognizing 14 categories, we need to construct a dataset of training images that includes samples from each category.

Data for the 14 categories were gathered by accruing images from ImageNet [13] and the Amazon Product Advertising API [2]. ImageNet is a widely used and comprehensive database that stores prelabeled images so it was a natural resource. The Amazon API was chosen because we wanted to build at least a modestly sized dataset without having to rely on only one data source. Since this API allowed for the automated retrieval of images from each specific category, the usually cumbersome data accumulation process was less so. Nonetheless, to ensure accuracy, we manually reviewed the entire dataset to ensure that each image was assigned an accurate ground truth label.

A total of 20,273 images were divided into 16,214 training and 4059 validation images. Further, 2,048 testing images were manually collected via smartphones by our team. To be viable, ESRS must be compliant with data inputed from mobile platforms. Thus, it made sense that smartphone images be used for judging the viability of the system. We wanted to ensure that authentic samples were taken to closely resemble the types of images that actual users of ESRS would likely capture. This is why the test dataset contains many low contrast images, a common problem when taking pictures with smartphones. This is a financially expensive and time consuming process as images need to be manually collected by individuals at locations that naturally contain sports and exercise equipment. For example, gyms, sports stores, sporting centers and private homes etc. Thus, the test dataset is quite small relative to the training dataset. Ideally, we would have liked to include mobile images within the training and validation datasets as well but this was not possible due to monetary and manpower constraints.

Table 1.1 provides a detailed numerical breakdown of the dataset and Figure 1.4 provides
Table 1.1: A numerical breakdown of all the categories and their respective train, validation and test sample sizes.

<table>
<thead>
<tr>
<th>Category</th>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Badminton Racquet</td>
<td>1152</td>
<td>288</td>
<td>138</td>
<td>1578</td>
</tr>
<tr>
<td>Barbell</td>
<td>1084</td>
<td>272</td>
<td>146</td>
<td>1502</td>
</tr>
<tr>
<td>Baseball Glove</td>
<td>1029</td>
<td>258</td>
<td>149</td>
<td>1436</td>
</tr>
<tr>
<td>Dumbbell</td>
<td>1407</td>
<td>352</td>
<td>193</td>
<td>1952</td>
</tr>
<tr>
<td>Golf Club Head</td>
<td>1073</td>
<td>269</td>
<td>110</td>
<td>1452</td>
</tr>
<tr>
<td>Kayak</td>
<td>901</td>
<td>226</td>
<td>102</td>
<td>1229</td>
</tr>
<tr>
<td>Road Bike</td>
<td>1241</td>
<td>311</td>
<td>193</td>
<td>1745</td>
</tr>
<tr>
<td>Running Shoe</td>
<td>1203</td>
<td>301</td>
<td>112</td>
<td>1616</td>
</tr>
<tr>
<td>Spinning Bike</td>
<td>1267</td>
<td>317</td>
<td>142</td>
<td>1726</td>
</tr>
<tr>
<td>Squash Racquet</td>
<td>1128</td>
<td>282</td>
<td>141</td>
<td>1551</td>
</tr>
<tr>
<td>Swimming Pool</td>
<td>919</td>
<td>230</td>
<td>160</td>
<td>1309</td>
</tr>
<tr>
<td>Table Tennis Racquet</td>
<td>950</td>
<td>238</td>
<td>124</td>
<td>1312</td>
</tr>
<tr>
<td>Tennis Racquet</td>
<td>1312</td>
<td>328</td>
<td>226</td>
<td>1866</td>
</tr>
<tr>
<td>Treadmill</td>
<td>1548</td>
<td>387</td>
<td>112</td>
<td>2047</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>16214</strong></td>
<td><strong>4059</strong></td>
<td><strong>2048</strong></td>
<td><strong>22321</strong></td>
</tr>
</tbody>
</table>

an example for each category found within the dataset. As was stated earlier, it was necessary to ensure that both resistance and aerobic training were covered due to their importance. Furthermore, we wanted to cover a wide variety of exercises and activities that people suffering from diabetes might partake in. This is a disease that disproportionately affects individuals 45 years and older [19]. Thus, we also ensured that at least some of the selected activities (e.g. golfing and swimming) were popular with people in this demographic.

1.4 Challenges

This work presents a set of challenges that must be overcome to develop a generalized and discriminative model capable of detecting sports and exercise equipment within natural images. In this section, we list the major challenges:

**Challenge I - Complex network vs. small training dataset:** The dataset consists of approximately 16 thousand training samples while our baseline architecture is comprised of more than 5 million parameters [59]. Given that the number of parameters is significantly greater than the number of training samples, it is expected that overfitting will be a major roadblock from the onset [54].

**Challenge II - Small inter-category and large intra-category variations:** Small inter-category variation exists between some of the categories. The Badminton Racquet, Squash Racquet and Tennis Racquet categories are very similar to each other since they are all different types of racquets and thus share a similar shape (see Figure 1.5). Tennis
Figure 1.4: The 14 different categories that ESRS is capable of recognizing.
1.4. Challenges

(a) Badminton Racquet  (b) Squash Racquet  (c) Tennis Racquet

Figure 1.5: 3 of the 14 categories are used in racquet sports and thus are very similar to each other in physical appearance. This similarity may cause confusion, especially when viewed from a side-on angle, for a classifier.

Racquets generally have larger heads while badminton racquets generally have smaller heads and distinguishably thinner frames. Furthermore, squash and tennis racquets are even more similar to each other than they are to badminton racquets. Generally, the distinguishing factor is the size and shape of the racquet head. However, images taken from a side-on angle can be especially problematic, as such an angle can conceal the main differentiating features. Furthermore, large intra-category variations exist within many categories. The dataset contains multiple items that vary significantly in appearance but would still be identified as being from the same category. Figure 1.6 provides examples from the Spinning Bike and Golf Club Head categories. This is a difficult challenge for any machine learning system.

Challenge III - Training and testing dataset variance: There are some fundamental differences between the training and validation sets versus the testing set. One of the primary distinctions is that the testing set consists of images captured via mobile phones especially for this study. Thus, it primarily contains the types of images that we hypothesize users will actually capture when using ESRS. On the other hand, the training and validation sets contain images that were extracted from sources on the Internet. Many of the images from these sets are not ideal for this study. For example, the training set contains images with annotations and animations which could mislead the classifier (see Figure 1.7). Furthermore, the testing set for the kayak category does not incorporate any examples of kayaks within natural bodies of water. This issue makes the kayak testing set quite different from the training and validation sets, as they both contain many images showing water bodies. Figure 1.8 provides examples. We explain the reason for these differences in Section 1.3.

Challenge IV - Humans and background clutter: All the equipment chosen as categories for this work have at least one thing in common: they were all designed for human beings. For example, barbells are used by people to perform resistance training exercises,
Figure 1.6: There is a considerable amount of visual variance that can exist within individual categories. We provide three examples each found within the dataset from the Spinning Bike (top row) and Golf Club Head (bottom row) categories to demonstrate the type of variance that exists. This intra-category variation poses a challenge as a classifier must be capable of recognizing the correct category despite the existence of subcategories within the dataset.

Figure 1.7: There is a disparity between the training and validation datasets in comparison to the test dataset. Many of the images within the former datasets contain annotations and animations. We have provided four examples from the dataset. This may mislead the classifier and teach it features that do no exist in the test dataset.
1.4. Challenges

Figure 1.8: The images found within the training and validation datasets for the Kayak category are very different than the ones found within the test dataset. The former datasets contain many examples of kayaks found within bodies of water (top row) while the latter dataset does not contain any depictions of bodies of water (bottom row). This disparity may make it difficult for a classifier to correctly identify kayaks within the test dataset.

Baseball gloves are worn by people to play Baseball, and treadmills are used by people for aerobic training purposes. Therefore, people are commonly embedded within many of the images in the dataset. Also, natural images tend to have large amounts of background clutter. This is especially true for this dataset as many of these categories will be found in a gym setting. Gyms tend to be somewhat chaotic places where multiple types of equipment can be found placed in the background. Figure 1.10 provides examples. The addition of humans and background clutter adds unnecessary noise, which poses a difficult problem for a machine learning system.

Challenge V - Multiple categories within individual images: Many images contain multiple categories (see Figure 1.9). The challenge here is that the classifier should be able to recognize the category which is the focal point of the image and ignore the secondary category in the background. For example, with Figure 1.9e the classifier should recognize this image as being part of the Treadmill category, which is the primary object, and ignore the barbell in the background. Figure 1.9d presents another tricky situation that is a common occurrence within the Kayak category’s dataset. Most kayak images contain depictions of naturally occurring bodies of water since kayaks are by design meant for traversing such terrain. The Swimming Pool category also inherently contain water and thus it may be challenging to distinguish between these two categories.

Challenge VI - Low contrast images: Smartphone cameras can produce an abundance of low contrast images. Such cameras are usually inferior to the equipment employed by professional photographers. Further, given that smartphones are ubiquitous in the modern world, the photographers themselves are frequently amateurs that tend to capture
1.5 Proposed Solutions

The challenges listed above imply that this object recognition problem would be difficult to solve algorithmically. Given the recent successes achieved by deep neural networks in the object recognition domain, we will tackle this problem with a learning based approach by using deep convolutional neural networks (CNN). Specifically, the architecture of GoogLeNet [59], winner of the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), will be used as a starting point. The experimental methodology will involve assessing each trained model according to the results achieved on the validation dataset. If a given model out competes other models on the validation dataset then it will be considered a candidate for further improvement. The final assessment will be made by testing the best performing models against the test dataset. The test dataset will not be used during training and validation so that we can determine whether our models are capable of generalization or not.
1.5. Proposed Solutions

Figure 1.10: All the categories found within this dataset have been designed for human use. Thus, naturally we find that many images within the dataset show humans manipulating an object (top row). Furthermore, natural images tend to contain a good amount of background clutter (bottom row) that adds noise to the input data. An ideal classifier needs to have the ability to disregard the human figures and background clutter.

For problems relating to overfitting (Challenge I), we propose applying regularization techniques. The literature presents multiple examples [58, 40] where regularization can help reduce overfitting and boost generalization.

The supervised learning approach of deep CNNs has achieved impressive results within the last few years [29, 59]. Nonetheless, a large amount of quality training data is still required in order to produce a discriminative classifier. To address small inter-category variations within the dataset (Challenge II), we have gathered an approximately equal number of samples for each of the three racquet categories. This helps to reduce the likelihood that the classifier will become biased towards one category over another. For Challenge II’s large intra-category variation problem, many different types of subcategories have been included within the dataset. Despite our solutions, this challenge remains an arduous problem for any recognition system to solve. The only robust approach involves expanding the dataset size significantly to include training images from every possible angle to address the inter-category issue and examples of every type of subcategory found within each category to address the intra-category issue.

The most straightforward solution for Challenge III also involves the construction of a large and high quality dataset. We need to include mobile images within the training set, not include images with annotations and animations, and to include images containing bodies of water for the Kayak category’s test dataset. Unfortunately, due to monetary and manpower limitations (see Section 1.3), the obvious solution was infeasible and our dataset is lacking in this regard. Thus, to bridge this disparity between mobile and non-mobile images, we propose a histogram transformation scheme that would be applied to the training set of crawled images. This transformation would seek to alter the histogram of each training image so that it has the same average distribution found within the set of mobile images.

The presence of humans and background clutter (Challenge IV) is unavoidable but their impact on the classifier can be minimized by including other images within the dataset that do
Figure 1.11: Smartphones tend to produce a relatively large amount of low contrast images as the devices are regularly used by amateurs in varying light conditions. Our test dataset contains many examples of such images. Such images can be difficult to classify as objects can easily be masked or hidden.

not include humans and clutter. This is why the training and validation datasets include images from the Amazon Product Advertising API [2]. This API specifically returns images that only include the object itself and is devoid of humans and background clutter. We have specifically designed the training and validation datasets in this manner so that the network can be exposed to examples with, and without, humans and clutter.

We propose to handle Challenge V in a similar manner to the ILSVRC competition. The existence of multiple categories is an unavoidable fact in natural images. Thus, ILSVRC uses the top-5 accuracy metric to determine whether the correct prediction has been obtained. In a similar vain, this work also proposes to include this metric when reporting results. However, the ILSVRC classification competition involves about 1000 categories while this work concentrates on classifying only 14 categories. Thus, examining both the top-1, top-3 and top-5 metrics will provide us with greater context so that accurate and unbiased conclusions can be reached.

We propose to tackle Challenge VI by performing color normalization and denoising operations to improve the contrast of an image with the goal of improving object recognition. Color normalization has previously been successfully used in object recognition tasks to improve contrast and reduce effects that cause occlusion [14, 18]. We propose a color normalization and denoising scheme that would be applied to each test sample before it is fed to the classification model. The aim is to preprocess images to reduce common occluding effects in smartphone images, such as low lighting.

1.6 Contributions

In this work, we propose a new system capable of detecting sports and exercise equipment within natural images. The system, known as the Exercise and Sports Equipment Recognition System (ESRS), is capable of recognizing 14 different categories. The system is trained using a deep convolutional neural network (CNN). Results show that our model improves on the state-of-the-art for this particular dataset. We go further by applying color normalization and denoising techniques to handle the presence of low contrast smartphone images. The following summarizes our main contributions:
• We present a novel recognition system, ESRS, capable of recognizing sports and exercise equipment within natural images that can be integrated with mobile devices to provide instant recognition.

• We demonstrate how ESRS improves on the state-of-the-art for this particular dataset. We show how our models significantly reduce overfitting and beat the baseline architecture’s top-1, top-3 and top-5 test accuracy, and overall network loss.

• We show that overfitting for models trained with complex network architectures and relatively low amounts of training data can be reduced by using L2 regularization and increasing dropout in the final fully connected layer.

• We introduce a new preprocessing scheme that employs color normalization and denoising techniques to improve overall prediction accuracy and the accuracy achieved on low contrast mobile phone images present within the test dataset.

1.7 Thesis Structure

The following describes how the rest of this thesis is structured:

• **Chapter 2** reviews the theory of artificial neural networks and the field of *deep learning* as it applies to object recognition.

• **Chapter 3** reviews the theory behind GoogLeNet, the methods employed to create a model with strong predictive power, and the respective results produced by our experimental models against the validation dataset.

• **Chapter 4** reviews color normalization and denoising theory.

• **Chapter 5** details the results garnered against the test dataset by the models that achieved the best performance on the validation dataset. We also present the results of preprocessing images with the color normalization and denoising techniques presented in the previous chapter.

• **Chapter 6** outlines our conclusions and discusses future work.
Chapter 2

Review of Artificial Neural Networks and Deep Learning for Object Recognition

Research on artificial neural networks started back in the late 1950s and early 1960s [47, 48, 49] but due to the lack of processing power required to train these networks, the research community eventually lost interest in them. Inevitably, hardware powerful enough to train complex networks became accessible. This advancement in hardware capability has led to the advent of highly complex and deep neural networks capable of achieving high levels of accuracy in many fields such as object recognition, speech recognition, text recognition and natural language processing. In particular, deep convolutional neural networks (CNN) have become very popular in the realm of object recognition. Today, they are the primary tool that researchers are focusing on to set the threshold for the state-of-the-art. Deep CNNs have won the classification part of the benchmark ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [52] from 2012 to 2014. Research into highly complex networks has been dubbed deep learning.

In this chapter, Section 2.1 introduces artificial neural networks; Section 2.2 covers the theory behind convolutional neural networks; and Section 2.3 details the recent advances in deep learning for object recognition. In the next chapter, we use our understanding of the deep learning theory explored in this chapter to train various deep CNN models and judge their performance according to the results achieved on the validation dataset.

2.1 Artificial Neural Networks

The concept of artificial neural networks (ANN) is loosely modeled around how the human brain performs computations. Any given ANN is constructed of a set of connected “neurons” or “processing units” [22], together they can be modeled as a graph. This software machine has the ability to gain knowledge by learning via examples and stores information in the form of values assigned to each neuron, called synaptic weights. A typical network consists of an input layer, \( n \) hidden layers and an output layer. Figure 2.1 illustrates the structure of a simple ANN.

Typically, during the learning process, a network’s weights are systematically manipulated to fulfill a well described objective. For example, a binary task of classifying a set of images as
containing human faces or not. This controlled manipulation operation, known as the *learning algorithm*, aims to adjust the weights to improve the overall performance of the network in regards to the problem that is being tackled. But during the learning phase, the network can only be shown a finite number of examples. The real challenge comes when previously unseen examples are presented. If our network is as good at handling new examples as it was at handling training examples then we can say that a general model has been developed.

![Figure 2.1: A simple artificial neural network that is fully connected. Every neural network has one input layer, one output layer and at least 1 hidden layer. It is called a fully connected network because each neuron in layer $l$ connects to every neuron in layer $l+1$.](image)

### 2.1.1 Perceptrons

The perceptron is a type of artificial neuron that was introduced in [47, 48, 49]. It is not commonly used in modern ANNs but perceptrons are a good stepping stone towards gaining a more in-depth knowledge of how ANNs function because they are noted to be the simplest form of a neuron.

This simple processor receives multiple inputs $x_1, x_2, x_3, \ldots, x_m$ and outputs a binary result (see Figure 2.2). Each input is assigned a weight $w \in \mathbb{R}$, representing the level of importance that that input item holds. The output, 1 or 0, is calculated by applying an *activation function*, which in the case of perceptrons, is normally just a simple threshold calculation. Equation 2.1 showcases this activation function:

$$v = \sum_{i=1}^{m} w_i x_i + b_i$$

$$\text{output}(v) = \begin{cases} 1, & \text{if } v > 0 \\ 0, & \text{otherwise} \end{cases}$$

(2.1)

$b_i$ is known as the neuron’s bias and $m$ is the total number of inputs. The bias impacts the likelihood of the neuron returning a 1 (also known as *firing*). That is, the larger the bias, the higher the probability that the neuron will fire. It alters the output by applying an *affine transformation*, which can help in finding a better fit for our data.
2.1.2 Sigmoid Neurons

Sigmoid neurons are more commonly used than perceptrons because their output is not simply binary. Instead of producing just a value of 0 or 1, their values can be any number $x$ that satisfies $0 \leq x \leq 1$. Figure 2.3 illustrates the difference. The perceptron essentially employs a step function while the sigmoid function is a smooth curve. This property allows networks comprising sigmoid neurons to make small iterative changes. That is, a small change in the input leads to a small change in the output. This property makes these neurons more desirable to the learning process when compared to perceptrons as even minute changes can quite often cause the perceptron to be turned on or off thus leading to a very different output. The following is the sigmoid function:

$$\sigma(v) = \frac{1}{1 + e^{-v}} \quad (2.2)$$

Another advantage is that the sigmoid function (Equation 2.2), also known as the logistic function, is differentiable while the perceptron’s threshold function (Equation 2.1) is not [22]. Differentiability is an important property for the learning process as will be shown when we outline the backpropagation algorithm in Section 2.1.4.

2.1.3 Multilayer Perceptrons

Multilayer perceptrons (MLP) are a type of feedforward neural network. Despite the name, they are comprised of multiple layers of sigmoid neurons (see Section 2.1.2) connected together in order to feed information in one direction, forward, from the input layer to the output layer while passing through $k$ hidden layers (see Figure 2.4) where $k \geq 1$. Such networks can be trained in a supervised manner by using the backpropagation algorithm (see Section 2.1.4) [50] in combination with a gradient descent method to minimize the network’s learning error (also known as loss or cost) in order to improve the performance of the network. They can be modeled as directed acyclic graphs and have the capability to solve nonlinear problems [12].

Even though MLPs are a type of feedforward network, the learning process can be divided into two periods: the *forward pass* and the *backward pass*. First, during the forward pass,
2.1. Artificial Neural Networks

Figure 2.3: A comparison of activation functions. The perceptron activation function can only produce an output of 0 or 1 while the sigmoid activation is smoother as it allows any value between 0 and 1. This smoothness is a desirable trait for a neuron as small changes in input weights lead to small changes in the output. In contrast, small weight changes to the input of a perceptron function can lead to large changes in the output.

neuron weights remain the same, and the input signal travels from the input layer to the output layer as visualized by Figure 2.5. The results provided by the output neurons are then interpreted to determine whether the network was successful in solving the problem being tackled. Subsequently, during the backward pass, the output of the forward pass is used to calculate the network’s error rate by comparing the output against the ground truth. Then an error signal is propagated backwards through the network (see Figure 2.5) so that the weights of each neuron can be adjusted in order to reduce the network’s error rate during the next forward pass. Thus, in any given MLP a neuron, \( u \), must perform two tasks:

1. calculate the input signal by using its weights \( w_u \) and biases \( b_u \)
2. calculate the gradient of the error function required by the backward pass to locate the error’s global minima (see Figure 2.6)

During the forward pass, as the input signal travels deeper through the hidden layers, the vectors produced by nonlinear activation functions of each neuron act as feature detectors. Each progression through a set of hidden neurons produces more discriminative features that have been transformed into a higher dimensional space. This is done as such a space may provide a more straightforward way of separating the data into a distinguishable set of boundaries.

2.1.4 Backpropagation and Gradient Descent

A network’s learning error or loss can be formalized with a loss function. Backpropagation [51, 28] is an algorithm that can calculate the gradient of this function. This gradient is then
supplied to an optimization method known as gradient descent which adjusts the weights and biases of the network to find the global minima of the loss function (see Figure 2.6).

Given the loss function, $E$, backpropagation works by calculating the partial derivatives $\frac{\partial E}{\partial w}$ and $\frac{\partial E}{\partial b}$ for any weight $w$ or bias $b$. These two parameters have a direct impact on $E$ and thus if their rate of change can be quantified then they can be adjusted in a manner such that the loss is minimized. This is achieved by first computing the loss produced by the output layer and then working backwards on a layer-by-layer basis until the start of the network is reached. Algorithm 1 outlines the steps involved in determining the gradient of $E$ for a given training sample, $s$, and its corresponding ground truth label, $g$.

**Algorithm 1 Gradient of the Loss**

1: function LossGradient($s$, $g$)
2: Forward propagate the inputs for each layer, $l$ to compute the output, $p^L_i$
3: Compute the output loss, $E$
4: Backpropagate $E$ from $L, L - 1, L - 2, ..., l = 0$ to compute the loss for each layer, $E^l$
5: Compute the gradient for $E$: $\{\frac{\partial E}{\partial w}, \frac{\partial E}{\partial b}\}$
6: return $\{\frac{\partial E}{\partial w}, \frac{\partial E}{\partial b}\}$
7: end function

To understand backpropagation, let us first closely examine how forward propagation functions (line 2 in Algorithm 1). In an $L$ layer network, $N$, the end product is the output produced by the $i$th neuron in the $L$th layer, $p^L_i$. In order to garner the final output, the network must first perform the required intermediate arithmetic by determining the output of all neurons in the hidden layers. Equation 2.3 showcases how the output of the $i$th neuron, $u^l_i$, found in the $l$th
2.1. Artificial Neural Networks

Figure 2.5: Neural network learning has two stages of processing. The forward pass determines the network’s output. The backward pass first calculates the overall error and then uses backpropagation and gradient descent to update the weights to minimize this error. These two stages are repeated until the error is minimized and the best possible performance is achieved.

Figure 2.6: The surface area of a loss function with a local minima and a global minima. Backpropagation identifies the gradient while gradient descent traverses the surface area to locate the global minima.

Layer can be calculated:

\[ q^l_i = \sum_{j=1}^{j} w_{ij}^l p_{j}^{l-1} + b_i^l \]

\[ p_i^l = a(q_i^l) \] (2.3)

where \( q_i^l \) is the weighted input and \( a \) is a nonlinear activation function that is differentiable. For example, \( a \) could be the sigmoid activation function (see Section 2.1.2).

This equation demonstrates that the output of a neuron is dependent on the output of all neurons in the previous layer. Given the outputs of the previous layer, a weighted sum is performed, the bias is added and finally a nonlinear activation function is applied to achieve the final result.

The next step involves determining \( E \) for \( N \). There are many different loss functions available for use to quantify the loss, such as: cross entropy [31, 60] or the simple mean squared error:
Figure 2.7: The notation $w_{ij}^l$ and $p_i^l$ is used to identify weights and outputs respectively. Such a weight connects the $i^{th}$ neuron in the $l^{th}$ layer to the $j^{th}$ in the $(l-1)^{th}$ layer.

$$E = \frac{1}{n} \sum_{k=1}^{n} (g_k - s_k)^2$$  

(2.4)

where $n$ denotes the total number of training samples, $s_k$ is the network’s prediction for the training sample and $g_k$ is the corresponding ground truth label.

For the backward pass, the first operation involves calculating the loss in the output layer, $\partial E / \partial q_i^L$, via the chain rule [3, pp. 174 – 179]:

$$\frac{\partial E}{\partial q_i^L} = \sum_{j=1}^{J} \frac{\partial E}{\partial p_i^L} \frac{\partial p_i^L}{\partial q_i^L}$$

(2.5)

Recall $p_i^L = a(q_i^L)$ from Equation 2.3, which gives us:

$$\frac{\partial E}{\partial q_i^L} = \frac{\partial E}{\partial p_i^L} a'(q_i^L)$$

(2.6)

Next, the loss in any layer $l$ is determined by taking into account the error in the next layer. Again, employing the chain rule we get:

$$\frac{\partial E}{\partial q_i^L} = \sum_{j=1}^{J} \frac{\partial E}{\partial q_j^{l+1}} \frac{\partial q_j^{l+1}}{\partial q_i^L}$$

(2.7)

Note that the weighted input for the next layer, $q_j^{l+1}$ can be expressed as:
2.1. Artificial Neural Networks

\[ q_{j}^{l+1} = \sum_{j=1}^{j} w_{ji}^{l+1} p_{i}^{l} + b_{j}^{l+1} \]

\[ = \sum_{j=1}^{j} w_{ji}^{l+1} a(q_{i}^{l}) + b_{j}^{l+1} \]  

(2.8)

Differentiation provides us with:

\[ \frac{\partial q_{j}^{l+1}}{\partial q_{i}^{l}} = w_{ji}^{l+1} a'(q_{i}^{l}) \]  

(2.9)

And the loss of the \( i^{th} \) neuron in the \( l^{th} \) layer can be formalized as:

\[ \frac{\partial E}{\partial q_{i}^{l}} = \sum_{j=1}^{j} \frac{\partial E}{\partial q_{j}^{l}} w_{ji}^{l+1} a'(q_{i}^{l}) \]  

(2.10)

By starting at the \( L^{th} \) layer and working backwards \( L, L-1, L-2, ..., l = 0 \), Equations 2.6 and 2.10 can be combined to ascertain the error for each neuron in the network.

Now we can examine how the loss is changing in relation to any given \( w \) in the network, which is formalized by:

\[ \frac{\partial E}{\partial w_{ij}^{l}} = \frac{\partial E}{\partial q_{i}^{l}} p_{j}^{l-1} \]  

(2.11)

Lastly, the following is how the loss changes in relation to any \( b \) in the network:

\[ \frac{\partial E}{\partial b_{i}^{l}} = \frac{\partial E}{\partial q_{i}^{l}} \]  

(2.12)

Since \( \frac{\partial E}{\partial q_{i}^{l}} \) (see Equation 2.10) is already known, \( \frac{\partial E}{\partial w} \) and \( \frac{\partial E}{\partial b} \) can be derived. Given the above details, Algorithm 1 can be rewritten as shown by Algorithm 2 below.

Having determined the gradient of the loss, the weights can be subsequently updated by applying the following functions:

\[ w_{ij}^{l} \leftarrow w_{ij}^{l} - \eta \frac{\partial E}{\partial w_{ij}^{l}} \]  

\[ b_{i}^{l} \leftarrow b_{i}^{l} - \eta \frac{\partial E}{\partial b_{i}^{l}} \]  

(2.13)

where \( \eta \) is the learning rate parameter that dictates the amount by which the weights and biases are modified.

Note that so far, we have only discussed how to compute the gradient of the loss for a single training sample. But in any real scenario one will have many training samples. We can deal with this issue by employing stochastic gradient descent, which can calculate the gradient for many samples. Algorithm 3 demonstrates.
Algorithm 2 Gradient of the Loss (detailed)

1: function LossGradient(s, g)
2: Forward propagate the inputs for each layer, $l$ to compute:
3: \[ q_i^l = \sum_{j=1}^{J} w_{ij}^l p_{j}^{l-1} + b_i^l \]
4: \[ p_i^l = a(q_i^l) \]
5: Compute the output loss:
6: \[ \frac{\partial E}{\partial q_i} = \frac{\partial E}{\partial p_i} a'(q_i) \]
7: Backpropagate $E$ from $L, L-1, L-2, ..., l = 0$ to compute the loss for each layer:
8: \[ \frac{\partial E}{\partial q_i} = \sum_{j=1}^{J} \frac{\partial E}{\partial q_i} w_{j}^{l+1} a'(q_i^l) \]
9: Compute the gradient for $E$:
10: \[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial q_i} p_i^{l-1} \]
11: \[ \frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial q_i} \]
12: return \{\(\frac{\partial E}{\partial w_{ij}}, \frac{\partial E}{\partial b_i}\)\} $\triangleright$ Quantifying the loss gradient for the network
13: end function

Algorithm 3 Stochastic Gradient Descent

1: function SGD(samples) $\triangleright$ a mini-batch of $n$ training samples
2: for each $s, g$ in samples do
3: \{\(\frac{\partial E}{\partial w_{ij}}, \frac{\partial E}{\partial b_i}\)\} $\leftarrow$ LossGradient(s, g) $\triangleright$ see Algorithm 2
4: \[ w_{ij}^l \leftarrow w_{ij}^l - \eta \frac{\partial E}{\partial w_{ij}} \] $\triangleright$ update all weights
5: \[ b_i^l \leftarrow b_i^l - \eta \frac{\partial E}{\partial b_i} \] $\triangleright$ update all biases
6: end for
7: end function

2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are a type of feedforward network that have been applied to fields such as computer vision [10, 29, 55, 33], natural language processing [25, 27, 56, 61], speech recognition [1] and document recognition [32]. In the vision domain, they are designed to handle raw image data as input and identify patterns with minimal preprocessing and possess strong invariance to distortions and transformations. In essence, they are very good at understanding the contents of natural images.

Further, CNNs have fewer connections and parameters when compared to standard feedforward networks of similar width and depth and thus are easier to train [29]. A CNN is made up of a combination of convolutional, subsampling, activation and fully connected hidden layers along with the basic input and output layers. The brunt of the processing is done in the convolutional layers. These layers use convolution operations to produce feature maps, which act as the high level features required to complete the task at hand, such as object classification or detection. In the following sections, we first discuss how convolution works mathematically and then detail each type of layer followed by a review of architectures traditionally employed in CNN design.
2.2.1 2D Convolution

In image processing, 2D convolution involves sliding a filter (also known as a mask or kernel) across the width and height on an input image. The product (also called a feature map in the context of CNNs) is a new image that has been altered to induce a specific response such as highlighting edges, blurring and sharpening. Figure 2.8 shows how a filter can be applied for edge detection.

Let $I[a, b]$ denote a grayscale matrix that represents an input image and $F[q, r]$ denote a matrix that represents a filter that is convolved on the image. We can then define the discrete convolution operation by the following:

$$C[x, y] = \sum_{c=0}^{a-1} \sum_{d=0}^{b-1} I[c, d] * F[x - c, y - d]$$

(2.14)

It is important to note how this operation is applied when the filter extends beyond the boundaries of the image. This situation can be handled in the following ways:

- Duplicating - Extend the image by duplicating pixels at the edges.
- Cropping - Pixels that would require manufactured values to complete the convolution process are cropped. This leads to a smaller output image.
- Wrapping - Missing pixel values are taken from the opposite side of the image.

In the next few sections, we present the layers typically employed by CNNs. These layers act in a systematic fashion to transform the input image into an output that can be used for decision making.

2.2.2 Input Layer

The initial layer functions by receiving raw images with three channels of color (RGB). These images are then optionally cropped, rotated and transformed before being supplied to the first hidden layer.
2.2.3 Convolutional Layer

A convolutional layer convolves a set of filters over the entire surface area of an input image to produce a set of feature maps. The contents of these filters form the weights of the neurons in the layer and are learned during training such that the feature maps produced prioritize discriminative features within the input data. Initial layers might learn basic features such as edges, contours and textures while deeper layers identify higher level features that inherently describe the image [64, 63]. For example, if the picture of a road bike is presented to a CNN then these higher level features might be the wheels, handle or seat of the bike.

Unlike an MLP, the neurons in a convolutional layer do not connect to all neurons from the previous layer. Instead, each neuron only connects to a certain subset of neurons from the previous layer. Such layers are considered to be locally connected while layers in an MLP are fully connected (see Figure 2.9). For the first convolutional layer, this property has the implication of having each neuron only cover a certain spatial region of the input image, called the local receptive field. This is done so that spatially local patterns produce strong activations. Furthermore, images are high dimensional structures and using fully connected structures can lead to untenable computational complexity and a slower convergence to a global minima. Thus, to maintain achievable training speeds, the amount of connectivity between neurons is reduced. Figure 2.10 visualizes the filters and output of the first convolutional layer of a CNN.

Weight sharing is another method employed to reduce complexity by reducing the number of parameters within the network. For each filter in a given layer, the weights remain the same when sliding across the entire surface area of the input. This allows each filter to specialize in detecting a particular feature across the entire input. This approach is employed because if a particular structure exists in a certain local patch then there is a high probability that it might also be found in other patches of the input. Essentially, this trick makes CNNs translation invariant and thus they are more robust at handling natural images.

Each convolutional layer requires the following hyperparameters:

- Number of filters - Defines the number of filters convolved over the input for a particular
2.2. Convolutional Neural Networks

- Size of each filter - Defines the size of the filter used during the convolution process.
- Stride - Dictates by how much a filter moves across the width and height dimensions of the input during the convolution process. For example, a stride of 1 means that the filter moves across the image 1 pixel at a time.
- Padding - Allows for the padding of the input with zeros. This allows us to control the size of the output.

![Input Image](a) Input Image  ![First Convolutional Layer](b) First Convolutional Layer  ![Filters](c) First Convolutional Layer  ![Output](d) First Convolutional Layer

**Figure 2.10:** A visualization of an image from the BaseballGlove category being processed by a modified version of GoogLeNet (Model I). (a) The input image supplied to the CNN (b) Visualization of the filters produced by the first convolutional layer (c) Visualization of the output produced by the first convolutional layer.

We can now take into account Equation 2.14 to re-express Equation 2.3 (the output of a layer, $l$) in terms of a CNN:

$$q_i^l = \sum_{j=1}^{h_{l-1}} w_{ij}^l * m_{ji}^{l-1} + b_i^l$$

$$m_i^l = a(q_i^l)$$

(2.15)

where $m_i^l$ is the $i$th feature map for the $l$th layer, $w_{ij}^l$ is the weights used as a filter, and $h_{l-1}$ is the count of feature maps from the previous layer. Note that the activation function, $a$, is usually applied by CNNs in a separate activation layer.

2.2.4 Subsampling Layer

Subsampling or pooling layers are usually applied immediately after convolutional layers. They are responsible for extracting only the most relevant data from the feature maps produced by the previous layer. This procedure again has the benefit of reducing the number of
parameters within the network, which has various performance related benefits as discussed in the previous section. Further, pooling helps improve translation invariance and reduce the impact of noise [6]. The following are two commonly used pooling operations:

- **Max Pooling** - This procedure involves sliding an $n \times n$ filter with stride $s$ across the entire surface area of a feature map. The filter acts on the feature map by extracting the maximum value at each $n \times n$ patch to eventually produce a subsampled output (see Figure 2.12). This method has proven to be helpful in increasing training speed [53]. If $n > s$ then overlapping pooling is performed as the filters are applied in overlapping patches across the input. This slight modification has been shown to reduce overfitting [29].

- **Average Pooling** - Similar to max pooling, this operation slides a filter across a feature map. However, instead of using the maximum value for a given patch, the average value is calculated (see Figure 2.12). This procedure can be helpful in handling foreground variance within the input [5].

2.2.5 Activation Layer

This is the layer that applies nonlinearity to the input via an activation function. The rectified linear units (ReLU) function has now become the activation method of choice for many CNN implementations. Other types of activation functions, such as tanh or sigmoid, have been used previously but employing ReLU can result in an increase in training speeds [29]. We can formalize ReLU with the following equation:

$$v = wx + b$$
$$\text{output}(v) = \max(0, v)$$ (2.16)

2.2.6 Fully Connected Layer

Neurons in a fully connected layer match the structure found in a MLP layer. That is, instead of being connected to a subset of neurons from the previous layer, each neuron is connected to
2.2. **CONVOLUTIONAL NEURAL NETWORKS**

![Diagram of pooling operations](image)

**Figure 2.12:** Two common pooling operations used in subsampling layers are depicted. We show the result of sliding a $2 \times 2$ filter across a $4 \times 4$ feature map. Max pooling takes the maximum value found within the $2 \times 2$ patch while average pooling takes the average of the values within the patch.

All neurons in the previous layer (see Figure 2.4). Typically, this layer is found after a series of convolutional, pooling and activation layers. Its main job is to aggregate all the data across the entire input set to examine it from a global perspective, instead of focusing on spatial observations found at a local level.

### 2.2.7 Loss Layer

In this layer, the ground truth is compared against the prediction made by the network to determine how well the network is performing. The loss is calculated to quantify network performance with the ultimate goal of minimizing the loss via backpropagation and gradient descent (see Section 2.1.4). The softmax loss function is a commonly used method for multi-category classification purposes [29, 59]. Formally, the softmax loss for the output layer:

$$p_i^j = \frac{e^{q_i^j}}{\sum_{j=1}^{J} e^{q_j^j}}$$

(2.17)

where $q_i^j$ signifies the weighted input and $p_i^j$, the corresponding output (see Equation 2.3). One of the great advantages of using softmax as the classifier is that it provides a prediction probability distribution that adds up to 1 for the entire set of categories. Thus, we can judge the strength of the classification by the assigned probabilities. For example, if category A is assigned a 0.95 probability then we can say that the network is very certain that the sample being processed belongs to category A, while if the probability was merely 0.55 then we would assert that the network is relatively uncertain.
2.2.8 Traditional Architectures

There are no hard set rules on how one should design a CNN architecture. Instead, there are general guidelines and best practices that a designer can use along with expert knowledge of the application domain to build an architecture that performs well for a given dataset. Typically, the following three layers are grouped together in the order displayed:

1. Convolutional layer
2. Subsampling layer
3. Activation layer

This group, which we call the convolutional group, is then repeated throughout the network with differing hyperparameters. Lastly, we find that a fully connected layer is typically added right before the final loss layer. [32] defines a similar architecture but only uses one convolutional group. [29] is also very similar, except it is significantly deeper with multiple convolutional groups spread throughout the network.

2.3 Recent Advances in Deep Learning for Object Recognition

Over the years, the literature has presented a plethora of machine learning techniques to solve the object recognition problem. For example, methods such as AdaBoost [62, 34] and Support Vector Machines [17, 39] have been used to extract good results. However, today the field of computer vision is experiencing a virtual revolution brought forth by deep CNNs. In the last few years, researchers have used deep CNNs to achieve substantial advances in benchmark classification and localization tasks as well as real world applications.

[29] is a landmark paper by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton that pushed the field significantly forward by achieving groundbreaking results in the 2012 edition of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [52]. Their network procured a top-5 accuracy of 84.7%, which bested the second place classifier by 10.9%. Many papers in the field have subsequently expanded the literature by using their work as a foundation. They employed backpropagation and gradient descent based learning via GPUs on 1.2 million training images accompanied by image cropping, L2 regularization and dropout strategies (see Section 3.4) to limit overfitting.

ILSVRC 2014 witnessed another substantial increment in the state-of-the-art. Again working with a 1.2 million large ImageNet dataset, GoogLeNet [59], devised by a team from Google (as the name suggests), achieved a top-5 accuracy of 93.3% in the classification challenge, a 8.6% improvement on [29]. The network also came out on top in the object detection challenge. Their main contribution was the introduction of a new type of layer organization paradigm, called the Inception module (see Section 3.3). [52] compared GoogLeNet’s performance with that of two expert annotators and found that the network achieved comparable performance. The first expert outperformed GoogLeNet by 1.7% while the second expert underperformed by 6.2%. The experts were only able to achieve similar performance rates after a significant
amount of practice and their labeling rate was only about 1 image per minute (note that some images were identified quicker than others).

Real world applications by industry have also started to take root. In 2013, a team at Google devised a system [21] trained via deep CNNs to recognize numbers from images found in the Google Street View application. Their system showcased human level accuracy by achieving accuracy rates above 90% combined with inhuman levels of speed. For example, their system was able to identify all street numbers in France in under 1 hour. Another application includes employing deep CNNs to detect cancer causing cells [24] by examining biological tissue. This team trained two deep CNNs to distinguish between malignant and benign forms of colorectal cancer. Using a set of 125 thousand training images, they were able to achieve accuracy rates of 98% and 94% for two different test sets.

Deep learning continues to show promising results in object recognition and in a variety of different fields such as natural language processing and speech recognition. Having said that, it is important to note that our understanding of deep CNNs is still in its infancy. More research is required to fully understand why deep CNNs perform so well on object recognition and other tasks.
Chapter 3

Training Deep Networks for Exercise and Sports Equipment Recognition

The progress in object recognition via deep learning is very promising, as we outlined in Section 2.3. Our methodology, presented in this chapter, attempts to build on previous work to achieve the best possible results in exercise and sports recognition. In particular, GoogLeNet [59] is used as a starting point to train a baseline model (see Section 3.1) for this dataset. We then train a set of models via hyperparameter tuning (see Section 3.2), architecture augmentation (see Section 3.3), regularization (see Section 3.4) and transfer learning (see Section 3.5) to experimentally judge which model produces the best results in comparison to the baseline. This is assessed by comparing the performance achieved by each model against the validation dataset (see Section 3.6 for a summary of results). Along the way, we also present the theory behind GoogLeNet, regularization and transfer learning.

In Chapter 5, the models that performed the best during validation are then tested against the test dataset. Also, given the presence of low contrast images within the test dataset, we apply color normalization and denoising techniques to see whether accuracy rates can be further improved. The theory behind these techniques is reviewed in Chapter 4.

3.1 Training a Baseline Model

The results achieved by deep CNNs over the past few years have been extraordinary. Thus, we made the decision to build on this work by using a deep CNN to tackle the task of recognizing exercise and sports equipment in natural images, with GoogLeNet as the selected baseline architecture. The reason for this is two fold. Firstly, this work also partly operates on images from ImageNet (see Section 1.3). GoogLeNet has been shown to perform well with this dataset. Secondly, this network has demonstrated state-of-the-art performance in both object classification and detection tasks (see Section 2.3), with accuracy rates matching what human experts were able to achieve on the same dataset. Thus, the initial objective is to create a baseline model, designated Model A, for our dataset by training GoogLeNet as presented in [59].

In order to train the baseline (Model A) and all other models in this work, a C++ based deep learning framework called Caffe [23] and an Nvidia GeForce GTX 980 GPU were deployed.
### 3.2. Hyperparameter Tuning

Table 3.1: Initial hyperparameter selections for the baseline model (Model A).

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>$1.0 \times 10^{-2}$</td>
<td>The learning rate</td>
</tr>
<tr>
<td>Learning Policy</td>
<td>poly</td>
<td>The learning rate follows a polynomial decay pattern</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
<td>The momentum used for gradient descent</td>
</tr>
<tr>
<td>Gradient Descent Type</td>
<td>Stochastic Gradient Descent</td>
<td>The type of gradient descent used during learning</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>$2.0 \times 10^{-4}$</td>
<td>Regularization parameter</td>
</tr>
<tr>
<td>Test Interval</td>
<td>4000 iterations</td>
<td>Interval between testing the model’s performance with the validation data</td>
</tr>
<tr>
<td>Test Iterations</td>
<td>1000 iterations</td>
<td>Number of iterations to perform for each test attempt</td>
</tr>
</tbody>
</table>

Table 3.2: Results for Model A (baseline) garnered by training the standard GoogLeNet architecture. These results are used as a benchmark to measure the success of all other models. Note: The Loss Difference column indicates the difference between the train and validation loss.

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Loss: $1.87 \times 10^{-2}$</td>
<td>Min Loss: 0.865</td>
<td>Max Top-1: 81.9%</td>
</tr>
</tbody>
</table>

Table 3.1 presents the initial hyperparameter selections.

Table 3.2 and Figure 3.1 present the results for Model A. We use these baseline results as a comparison benchmark to determine the success of all other models. The results show that overfitting is a problem and generalization is not being achieved due to the large difference between the train and validation loss. The relative percentage difference is 191.6%.

### 3.2 Hyperparameter Tuning

Table 3.1 lists a set of hyperparameters that we can tune in order to produce a more favorable output. Due to the relative simplicity, this is a common place to start when one is looking to improve the output model of a given architecture. During the weight and bias update process (see Section 2.1.4), the learning rate hyperparameter plays a significant role by dictating the size of the updates (see Equation 2.13). If the learning rate is too large then there is a good chance that the size of the updates will be too large, which means that we might “skip over” the global minima while descending the loss function’s gradient surface (see Figure 2.6). Alternatively, a small learning rate means that the weights and biases will be incremented by a small amount, which could increase the time needed to find the global minima. Thus, it is imperative that we identify a learning rate that is small enough to not “skip over” the global minima but large enough such that training time is minimized as much as possible.
Figure 3.1: Results for Model A (baseline). The large gap between the train and validation loss indicates that overfitting is occurring.

Table 3.3: Results for Model B. The same architecture as Model A is used but the learning rate hyper-parameter is reduced from $1.0 \times 10^{-2}$ to $1.0 \times 10^{-4}$.

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Loss</td>
<td>Min Loss</td>
<td>Max Top-1</td>
</tr>
<tr>
<td>$3.58 \times 10^{-3}$</td>
<td>0.771</td>
<td>81.4%</td>
</tr>
</tbody>
</table>

For Model B, we used the same architecture as Model A but reduced the learning rate to $1.0 \times 10^{-4}$ to ensure that the global minima was not being missed. The results (Table 3.3 and Figure 3.2) show that both the train and validation loss have been lowered in comparison to Model A. The top-1 and top-5 rates for both models remained about the same but it took Model B about 1500 more epochs to achieve the same rates. Most importantly, Model B’s results show that the problem of overfitting became worse as the percentage difference between the train and validation loss was 198.2%, a 6.6% increase over Model A. Given this increase in overfitting, despite the overall loss being reduced, we decided that it would be unviable to incorporate this model’s hyperparameter changes into future architectures.

3.3 Augmenting the Baseline Architecture

In this section, we first discuss in the detail the intuition and theory behind GoogLeNet’s architecture. Then we present the augmentations we made to the architecture in order to induce improved results for this dataset (see Section 1.3).
3.3. **Augmenting the Baseline Architecture**

![Image](image.png)

**Figure 3.2:** Results for Model B. The large gap between the train and validation loss indicates that overfitting is occurring.

### 3.3.1 The Intuition and Theory behind GoogLeNet

In order to understand GoogLeNet, we must first examine a particular structure that it uses extensively: Network in Network (NIN) [35]. The primary contribution of this paper involves the creation of what are known as micro networks. These networks can be stacked together in a deep network architecture to produce a more in-depth abstraction when compared to traditional CNN architectures. This is because CNNs are normally adept at working with data that is linearly separable and they produce generalized linear models (GLM). However, many problems are not linearly separable and the NIN architecture is good at discovering functions for nonlinear problems.

Each micro network, referred to as an mlpconv layer, is a convolutional layer that employs multiple fully connected layers to convolve over the input data. A set of back-to-back fully connected layers are otherwise known as an MLP. This MLP is connected to all local receptive fields within the input data to produce the output feature maps. We can interpret this procedure as an extra $1 \times 1$ convolutional layer with ReLU activation. The depiction in Figure 3.3 contrasts a traditional convolutional layer with an mlpconv layer. To summarize, this architecture has two main benefits: a greater capacity to identify distinguishing hyperplanes for complex data that is nonlinear in nature and the ability to reduce the complexity of the model by reducing the number of network parameters.

GoogLeNet, winner of ILSVRC 2014, builds on [35]'s micro network concept by introducing the Inception module [59]. Typically, deep architectures have a large computational budget and can be prone to overfitting due to the large number of parameters. Thus, this architecture aims to minimize the use of computing resources and limit overfitting for such networks by reducing the parameter count. Stacked layers of Inception modules are used along with dimension reduction strategies for efficient computation. The overall result is that deeper and wider networks can be built without the computations becoming increasingly untenable.

There are two straightforward ways to quickly improve the performance of a network:
increase the network’s depth and width, and increase the amount of training samples. However, both these approaches have their drawbacks. Deeper and wider networks can be prone to overfitting since there are more parameters to learn. This is especially true when training data is scarce. Unfortunately, increasing the size of the training data can also be a difficult task as in many cases humans with domain expertise may be required to accurately assign the ground truth labels. A further drawback of such large networks is the fact that they require significantly more computational power, a precious and finite resource, to execute.

The optimum solution to these drawbacks can be found by employing sparsely connected architectures rather than fully connected ones. This is because such sparsity not only resembles how biological neurons function but it also agrees with the theoretical work done by [4]. Their work states that one can construct the ideal sparse network structure by analyzing the statistics of neuron activations for each layer and accumulating together the neurons that fire simultaneously. This echoes the fundamental Hebbian principal from neuroscience - neurons that fire together, wire together. Unfortunately, today’s hardware is not proficient at handling sparse data structures that are non-uniform in nature. Given the limitations, structures need to be developed that can take advantage of the sparsity as per the recommendations of the theory but perform the numerical calculations on dense matrices to take advantage of current hardware. This proposed strategy led to the development of the Inception architecture.

The Inception architecture utilizes dense building blocks to emulate local sparse structures for a CNN aimed at solving problems in the computer vision domain. Every Inception module contains a $1 \times 1$ convolutional layer as per [35], a $3 \times 3$ convolutional layer, a $5 \times 5$ convolutional layer and a $3 \times 3$ max pooling layer positioned parallel to each other (see Figure 3.4a). A series of Inceptions modules are stacked on top of each other to form the main structure of a network. However, $5 \times 5$ convolutions can be very computationally taxing; when combined with pooling operations, the computations become untenable after just a few epochs of training. Thus, to deal with this issue a dimension reduction strategy of applying $1 \times 1$ convolutions prior to the $3 \times 3$ and $5 \times 5$ convolutions is employed to reduce the computational complexity (see Figure 3.4b). This alteration allows designers to create deeper and wider networks without concern for the computational costs.

GoogLeNet, depicted by Figure 3.5, is an instance of the Inception architecture with the specific goal of prioritizing computational efficiency. It has a depth of 27 layers. Overall, the entire depth and width of the network consists of 100 layers. The main concern with such a large structure is the ability to effectively backpropagate the gradients through the network. Thus, to mitigate this issue, the designers introduced two intermediate classifiers along with the final classifier at the end of the network. This was done to encourage the gradient signal, promote discrimination at the lower levels and increase regularization.

3.3.2 Architecture Augmentation Results

The first architecture augmentation we attempted (Model C) involved concatenating another Inception module at the very end of the architecture (see Figure 3.5) but before the final fully connected layer. The intuition behind this change was that deeper networks are better at understanding complex representations. The downside of adding another module, as stated in Section 3.3.1, is the increase in the computational cost. Since the hardware being applied to this problem remains unchanged, we expected that the training time required to minimize
3.3. Augmenting the Baseline Architecture

Figure 3.3: The mlpconv Layer, proposed by [35], employs multiple fully connected layers to convolve over the input data. It can be interpreted as an extra $1 \times 1$ convolutional layer with ReLU activation. Its benefits include reducing the number of network parameters and an improved ability to handle nonlinear data.

![Inception Module Structures](image)

(a) Initial Inception Module Structure   (b) Final Inception Module Structure

**Figure 3.4: Inception module structures (adapted from [59])**

The loss would increase. Nonetheless, given the relatively small training dataset, this was a compromise we were willing to make if the results improved in terms of accuracy and loss minimization.

The results for Model C, as per Table 3.4 and Figure 3.6, show that there was no improvement in terms of addressing overfitting. In fact, the results are inferior to the previous models as the difference between the train and test loss, 199.5%, actually increased. On the other hand, all the validation metrics are actually better than Model A, the baseline. Further, as predicted, this model took about 1000 more epochs to converge relative to Model A. Even though the validation metrics are an improvement over the baseline, we decided that this topology augmentation was defunct as overfitting worsened and the validation results were only slightly superior to the baseline.

The other architecture augmentation we attempted (Model D) involved adding another fully connected (FC) layer at the very end of the original network (see Figure 3.5). Fully connected layers are responsible for engaging in the high level reasoning within the network. [29] experimented with multiple fully connected layers at the end of their network and found that the results were positive. Thus, we wanted to empirically test whether adding another fully con-
Figure 3.5: The full GoogLeNet architecture (adapted from [59])
3.4 **Regularization to Reduce Overfitting**

The results we have detailed in the previous sections clearly show overfitting can be a major problem when training neural networks, especially when dealing with a relatively low amount of training data and a network that is quite deep and wide with a large amount of parameters to tune. This problem occurs when the model can readily identify most training data but fails to achieve the same rate of success when dealing with previously unseen data i.e. the validation and test data. That being said, there are some methods that have been identified in the literature to tackle this common problem. The easiest solution is to increase the size of the training dataset so that the model is exposed to more information, which improves its preparedness to understand previously unseen data. However, in many cases this is not a plausible solution. Research has revealed two effective regularization techniques to counteract overfitting: L2 regularization and dropout. In the next two sections, we discuss the theory behind both these techniques and the results we achieved by applying them to our problem.

![Train vs. Validation](image)

**(a) Train vs. Validation**

**Validation Accuracy**

![Validation Accuracy](image)

**(b) Validation Accuracy**

**Figure 3.6:** Results for Model C. The large gap between the train and validation loss indicates that overfitting is occurring.

**Table 3.5:** Results for Model D. The base GoogLeNet architecture is augmented by adding another fully connected layer at the very end of the network.

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Loss</td>
<td>Min Loss</td>
<td>Max Top-1</td>
</tr>
<tr>
<td>$1.14 \times 10^{-3}$</td>
<td>0.853</td>
<td>82.2%</td>
</tr>
</tbody>
</table>

A connected layer to our network would also improve the results for this dataset. However, the results (see Table 3.5 and Figure 3.7) show that this model performed at a similar level to Model C. The validation metrics improve upon the baseline model but overfitting became worse. Thus, we concluded that alternative methods would have to be explored.
3.4.1 Review of L2 Regularization

The idea of L2 regularization (also called weight decay) is based around expanding the original loss function by appending an additional term:

\[ E = E_{\text{orig}} + \frac{\lambda}{2} \sum_w w^2 \]  

(3.1)

where \( \lambda \) is the regularization parameter and \( \lambda > 0 \). Generally, L2 regularization discourages the formation of complex models by imposing a penalty on larger weights. The value of \( \lambda \) dictates the penalty applied to larger weights. The larger the value of \( \lambda \), the smaller the weights, and in turn, the more regularized the model will be. In a way, smaller weights are simpler and thus provide a less complex way of describing the underlying pattern within the data. By prioritizing smaller weights, we aim to build the smallest possible set of weights required to solve the problem and also limit the role of localized noise during the learning process [41, 40]. When the weights are relatively small, the network isn’t swayed by individual fluctuations in the input data. Instead, regularization promotes the recognition of the overall pattern found within the data. Finally, to incorporate this form of regularization into our network, we need to change the weight update method (Equation 2.13) used during the gradient descent process:

\[ w \leftarrow w - \eta \frac{\partial E_{\text{orig}}}{\partial w} - \eta \lambda w \]  

(3.2)

3.4.2 Review of Dropout

The basic idea of dropout [58] involves randomly excluding a certain percentage of neurons and their connections during a training pass (forward and backward pass). It is usually only applied to fully connected layers in a CNN. Figure 3.8 provides a depiction of a network with
3.4. **Regularization to Reduce Overfitting**

Figure 3.8: Dropout, a regularization procedure, works by deactivating a random subset of neurons within a network. In this depiction, three neurons have been disabled. This means that each input datum or each mini-batch is exposed to a slightly different network structure making it difficult for neurons to co-adapt.

dropout. This essentially translates to training each input datum or each mini-batch against a slightly different network structure. Taking this approach makes it difficult for neurons to co-adapt to each other when they can be randomly deactivated at any time, which compels each neuron to learn stronger features that describe the underlying pattern without relying on a specific set of other neurons [29]. In essence, this has a similar effect to L2 regularization. The network isn’t swayed as easily by localized noise since neurons that are being activated by the noise aren’t always active, which means their overall impact is greatly reduced and thus the task of overfitting to the training data becomes more difficult. However, note that during testing, the entire network is used. Since the network’s weights and biases were learnt during training when a certain percentage were not active, we must compensate by using smaller weights during testing. This is another way that dropout enacts regularization in a similar manner to L2 regularization.

We can take another approach to understand why dropout works so well in limiting overfitting in neural networks. Imagine if we could apply ten different neural networks to solve a problem rather than only one. We could ask our ten networks to tackle the problem and then use a democratic mechanism to make a decision. The different network topologies would converge and overfit in various different ways. If a majority of the networks, say eight out of ten, agreed, then we could be quite certain that the correct answer had been acquired. Now, normally using so many networks is not possible in the real world as there are usually various factors such as time and money that act as constraints. This is where dropout comes into the picture. Essentially, it allows us to artificially expose the training data to different network architectures and take into account the different ways in which each network overfits, which
Table 3.6: Results for Model E. We increase the regularization used by GoogLeNet by increasing the dropout ratio in the last fully connected layer from 40% to 55%.

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Loss</td>
<td>6.03 × 10^{-4}</td>
<td>199.7%</td>
</tr>
<tr>
<td>Min Loss</td>
<td>0.767</td>
<td>Max Top-1</td>
</tr>
<tr>
<td>Min Loss</td>
<td>97.5%</td>
<td>Max Top-5</td>
</tr>
</tbody>
</table>

Table 3.7: Results for Model F. We increase the regularization used by GoogLeNet by increasing the dropout ratio in the last fully connected layer from 40% to 70% and increasing the weight decay (L2 regularization) hyperparameter from 2.0 × 10^{-4} to 2.0 × 10^{-3}.

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Loss</td>
<td>0.420</td>
<td>49.0%</td>
</tr>
<tr>
<td>Min Loss</td>
<td>0.692</td>
<td>Max Top-1</td>
</tr>
<tr>
<td>Min Loss</td>
<td>96.7%</td>
<td>Max Top-5</td>
</tr>
</tbody>
</table>

can help in reducing overfitting.

3.4.3 Results from Applying Regularization

We created three different models to test the impact that regularization would have on our severe overfitting problem. The baseline model already implemented a dropout ratio of 40% in the final fully connected layer. In our first regularized model, Model E, we decided to increase this ratio to 55%. Table 3.6 and Figure 3.9 outline the results for this model. The loss, top-1 and top-5 metrics all improve on the baseline model but unfortunately the difference between the train and validation loss increased. The baseline recorded a percentage difference of 191.6% while for this model it was 199.7%. Thus, the gap grew wider, which leads us to conclude that this model overfits even more than the baseline.

Model F, our second attempt, combined an increase in dropout with an increase in the level of L2 regularization. The weight decay hyperparameter was increased to 2.0 × 10^{-3} from 2.0 × 10^{-4} and the dropout ratio in the final fully connected layer was increased to 70%. Table 3.7 and Figure 3.10 showcase the results for this model. One can clearly see that this change drastically curtailed the amount of overfitting. The gap between the train and validation loss for this model was only 49.0%, which is a significant improvement over the 191.6% difference achieved by the baseline model. Another point of interest is that the train loss is larger and the validation loss is smaller than the baseline. This is an indication that this model isn’t “memorizing” the traits of the training data unlike the previous models, all of which recorded a much smaller rate for the train loss. The last observation for this model is that the top-1 accuracy for the validation set, underperformed by 1.2% while the top-5 accuracy was about the same, in comparison to the baseline.

Our second attempt to reduce overfitting with Model F was a success but it was important to determine whether this was due to the increase in L2 regularization or the dropout ratio. Thus, for Model G, we kept the dropout ratio at 70% and reset the weight decay hyperparameter back to 2.0 × 10^{-4}. Table 3.8 and Figure 3.11 present the results. We observe that large amounts of overfitting is once again a problem with the gap between the train and validation loss going up to 199.7%. This result shows that the increase in L2 regularization played a major role.
3.5 Transfer Learning

Deep CNNs have the ability to learn primitive features (e.g. edges, contours, colors etc.) at lower layers and increasingly specialize, as would be expected, when moving to higher layers [63]. It has been documented within the literature that many features can be shared between...
such networks. These features can be transferred with positive results, such as a decrease in overfitting, if the problem sets are related at some level [64, 45]. For example, a CNN that specializes in classifying cars could be used to train a CNN that specializes in classifying sports cars. There are two practical transfer learning methods for neural networks:

**Finetuning weights** - This method uses a pretrained deep CNN as a weight initializer. The pretrained model is usually trained on a similar but larger training dataset. Then, instead of using random weights to initialize the network, we can use the weights from the pretrained model to “finetune” the network. This finetuning can be applied on a layer-
Table 3.9: To setup a benchmark, the baseline model (Model A) was finetuned. Table 3.10 and Figure 3.12 present the results for this finetuned model, called Model H. Our best performing model (Model G) was also finetuned so that it could be compared with the finetuned baseline. Table 3.11 and Figure 3.13 present the results for this model, called Model I.

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Original Model</th>
<th>Finetuned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Model A</td>
<td>Model H</td>
</tr>
<tr>
<td>Best ESRS Model</td>
<td>Model G</td>
<td>Model I</td>
</tr>
</tbody>
</table>

by-layer basis. Designers may sometimes opt to only finetune the higher level layers due to overfitting concerns.

Feature extractor - This method treats the deep CNN as a feature extractor. First, extract features from a convolutional layer of a CNN trained on a similar but larger dataset. Then use these extracted features to train a linear classifier such as a Linear Support Vector Machine for the target dataset, which is usually much smaller.

### 3.5.1 Results from Finetuning

In our experiments, we began with the feature extraction method by obtaining the features produced by the final convolutional layer and training a linear SVM. However, the results produced essentially matched our best performing models. Then we experimented with finetuning our best performing model (Model G) and the baseline. This is because a substantial portion of our dataset (see Section 1.3) has been acquired from the ImageNet database and GoogLeNet models, pretrained on the entire ImageNet database, are readily available for the Caffe deep learning framework [23].

The results show that both finetuned models easily outperform all the non-finetuned models. Furthermore, before finetuning was applied, the results indicated that Model G clearly outperformed the baseline model, Model A (see Section 3.4.3). However, after applying finetuning, we find that the performance achieved on the validation dataset is virtually the same. In fact, we find that the finetuned baseline performs slightly better in all metrics except for the minimum train loss. We also observe that the levels of overfitting pre and post finetuning remained unchanged but the training time required for the network to converge was significantly reduced. The finetuned version of Model A converged 63.4% faster than the original version, and the finetuned version of Model G converged 70.8% faster than the original version.

Table 3.10: Results for Model H. This is the finetuned version of the baseline model (Model A).

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Loss</td>
<td>Min Loss</td>
<td>Max Top-1</td>
</tr>
<tr>
<td>$5.08 \times 10^{-3}$</td>
<td>0.409</td>
<td>90.5%</td>
</tr>
</tbody>
</table>
**Figure 3.12:** Results for Model H. The large gap between the train and validation loss indicates that overfitting is occurring.

**Figure 3.13:** Results for Model I. The large gap between the train and validation loss indicates that overfitting is occurring.
Table 3.11: Results for Model I. This is the finetuned version of Model G.

<table>
<thead>
<tr>
<th>Train Min Loss</th>
<th>Validation Min Loss</th>
<th>Max Top-1</th>
<th>Max Top-5</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4.21 \times 10^{-3}$</td>
<td>0.421</td>
<td>90.3%</td>
<td>98.7%</td>
<td>196.0%</td>
</tr>
</tbody>
</table>

### 3.6 Summary of Validation Results

We used our dataset (see Section 1.3) containing images of exercise and sports equipment to train a baseline model (Model A) and eight different augmented models. Table 3.12 summarizes the training and validation results. As the table shows, each augmented model falls into a specific category (e.g. hyperparameter tuning, transfer learning etc.). Note that in our discussion below, we present the finetuned results separately from the non-finetuned results as the former relies on outside data.

The non-finetuned results show that Model G achieved the best minimum train loss of $5.76 \times 10^{-4}$, the best top-1 validation rate of 85.2% (along with Model E), and the best top-5 validation rate of 97.7%. The smallest loss difference, 49.0%, was achieved by Model F, which shows that this model was the most successful at limiting overfitting. This model also achieved the best minimum validation loss of 0.692. Overall, Model G was the clear winner as it outperformed the baseline in every metric, and all the other models in four out of five metrics. One important observation is that Model E achieved almost the same level of performance as Model G. Nonetheless, we designated Model G as the best non-finetuned model because it beat Model E in every metric and it achieved a better average top-1 accuracy rate of 81.0%, while Model E only achieved 80.8%. This metric is determined by measuring the top-1 validation accuracy every 4000 iterations during training.

The next step involved finetuning the baseline model and the best non-finetuned model, Model G. The results show that both models achieved almost the same level of performance in every metric. Although, the baseline model slightly outperformed its competitor in every metric except the minimum train loss. This is interesting because in the non-finetuned version of this comparison, Model G clearly outperformed the baseline model. Figure 3.12 shows the top-1 and top-5 metrics in a bar chart to clearly depict the best and worst models.
Table 3.12: A summary of results for the training and validation datasets. Note: The Loss Difference column indicates the difference between the train and validation loss.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Model Type</th>
<th>Train</th>
<th>Validation</th>
<th>Loss Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model A</td>
<td>Baseline</td>
<td>$1.87 \times 10^{-2}$</td>
<td>$0.865$</td>
<td>$81.9%$</td>
</tr>
<tr>
<td>Model B</td>
<td>Learning rate: $1.0 \times 10^{-4}$</td>
<td>$3.58 \times 10^{-3}$</td>
<td>$0.771$</td>
<td>$81.4%$</td>
</tr>
<tr>
<td>Model C</td>
<td>Extra inception module</td>
<td>$1.05 \times 10^{-3}$</td>
<td>$0.859$</td>
<td>$82.7%$</td>
</tr>
<tr>
<td>Model D</td>
<td>Extra FC layer</td>
<td>$1.14 \times 10^{-3}$</td>
<td>$0.853$</td>
<td>$82.2%$</td>
</tr>
<tr>
<td>Model E</td>
<td>Dropout: 55%</td>
<td>$6.03 \times 10^{-4}$</td>
<td>$0.767$</td>
<td>$85.2%$</td>
</tr>
<tr>
<td>Model F</td>
<td>Weight decay: $2.0 \times 10^{-3}$; Dropout: 70%</td>
<td>$0.420$</td>
<td>$0.692$</td>
<td>$80.7%$</td>
</tr>
<tr>
<td>Model G</td>
<td>Dropout: 70%</td>
<td>$5.76 \times 10^{-4}$</td>
<td>$0.771$</td>
<td>$85.2%$</td>
</tr>
<tr>
<td>Model H</td>
<td>Finetuned baseline</td>
<td>$5.08 \times 10^{-3}$</td>
<td>$0.409$</td>
<td>$90.5%$</td>
</tr>
<tr>
<td>Model I</td>
<td>Finetuned Model G</td>
<td>$4.21 \times 10^{-3}$</td>
<td>$0.421$</td>
<td>$90.3%$</td>
</tr>
</tbody>
</table>

Figure 3.14: The top-1 and top-5 accuracies achieved by each model on the validation dataset.
Chapter 4

Review of Color Normalization and Denoising to Improve Object Recognition

Low contrast images are a common problem for the average amateur photographer that uses a smartphone, especially in low-light conditions [44, 26]. Such images produce a high amount of noise, which can have a negative impact on classification accuracy. Our test dataset includes various low contrast images (see Figure 1.11) captured by different smartphones. In order to address this issue, and hopefully improve accuracy, we color normalize and denoise these test images in a preprocessing stage before submitting the images to our best performing models for classification. Color normalization encompasses a set of techniques to compensate for contrast variance within images due to differing light conditions and camera equipment for tasks such as object recognition or just to increase the overall visual quality of an image.

In this chapter, Section 4.1 reviews how images can be color normalized via histogram equalization techniques and Section 4.2 reviews the method we use to denoise images. We present our results from applying these techniques on the test dataset in Section 5.3. Note that color normalization and denoising are not applied to images within the training and validation datasets as they were not captured via smartphones and do not contain low contrast images. Section 1.3 provides more details on this difference between the training and validation datasets, and the test dataset.

Figure 4.1: The three different workflows we used for object recognition. The original workflow involves directly submitting the input image to the classification model for prediction. The other two workflows involve CLAHE and NL-means preprocessing.
4.1 Histogram Equalization

The Histogram Equalization technique examines the pixel intensity values as a histogram, a very common technique in computer vision. For grayscale images, it improves the contrast of the image by transforming the histogram values into a more uniform distribution by applying a cumulative distribution function (CDF) to the original histogram. See Figure 4.2 for an example. Thus, let CDF for a grayscale image be:

\[
\text{cdf}(j) = \sum_{i=0}^{j} H(i)
\]  

(4.1)

where \(H\) is the histogram for said image. Given cdf, the histogram equalization function can be formalized as:

\[
\text{histEqualize}(v) = \text{round}\left(\left(\frac{L - 1}{w \cdot h} \right) \frac{\text{cdf}(v) - \text{cdf}_{\text{min}}}{\text{cdf} - \text{cdf}_{\text{min}}}\right)
\]  

(4.2)

where \(v\) is the original pixel value, \(L\) denotes the possible pixel intensities, \(w\) the width of the image and \(h\) the height of the image.

For color images in the RGB space, this technique doesn’t work well when directly applied to all three dimensions. However, if the image is converted to the LHS or HSI spaces, one can then apply this method to only the luminance or saturation dimensions to get a similar result [43]. Overall, histogram equalization is ideal when pixel intensities are evenly distributed throughout the image. Alternatively, it does not perform well when the image contains regions that are very dark or light in comparison to the rest of the image.

![Figure 4.2: Histogram equalization transforms the pixel intensity histogram of an image so that the pixel intensities are more uniformly distributed by applying a cumulative distribution function (CDF) to the original histogram.](image)
4.1.1 Adaptive Histogram Equalization

Adaptive Histogram Equalization (AHE) is another equalization method that is designed to improve low contrast images. Unlike the conventional histogram equalization technique, it is adept at handling extremely dark or bright patches within an image. Instead of creating a histogram of the entire image, this method creates histograms for a square patch around each pixel (see Figure 4.3). Then each patch is histogram equalized as shown in the previous section. The size of the patch is a configurable hyperparameter. Larger patches reduce contrast while smaller patches increase contrast.

However, there are drawbacks, such as slow performance (time complexity of $O(n^2(m + L))$ for an $n \times n$ image and an $m \times m$ patch where $L$ denotes the possible pixel intensities) and the possible intensification of noise if the image contains swathes of regions with very little variance in intensity values [46]. Furthermore, aside from the ability to configure the patch size, this method does not provide an explicit way to control the level of contrast applied to an image.

![Figure 4.3: Adaptive Histogram Equalization functions by applying the histogram equalization technique on $n \times n$ patches throughout the image.](image)

4.1.2 Contrast Limited Adaptive Histogram Equalization (CLAHE)

The Contrast Limited Adaptive Histogram Equalization (CLAHE) method is an improvement on AHE (see Figure 4.4). It introduces the ability to control contrast and dissipate the enhanced noise that AHE normally produces [46]. For each image patch, we first check whether values in any bin of the histogram are above a certain contrast limit threshold, which is a configurable hyperparameter known as the *clip limit*. If so then there are two available methods to handle these overflowing values:

1. Non-uniformly transfer overflowing values to bins that do not exceed the clip limit: the main advantage to this approach is that contrast enhancement is applied to areas that need it the most. However, this can be a complex approach to implement as non-uniformly transferring pixels to other bins can change the overall intensity distribution of the image.
2. Redistribute overflowing values in a uniform manner to other bins: this method maintains the original overall intensity distribution of the image. However, this redistribution can cause other bins to overflow beyond the clip limit. In which case, if desired, this procedure can be repeated recursively until all bins are adequately below the clip limit.

In practice, given the complexity of the non-uniform approach, the second option of uniformly distributing the pixel intensities is generally preferred. Subsequent to the contrast limitation, CDF is calculated and the histogram is transformed for each patch in the image as described in the AHE section above.

### 4.2 Non-local Means Denoising

In terms of the noise produced, CLAHE is an improvement on AHE. Nonetheless, an explicit noise reduction method is also needed as the output produced by CLAHE can still contain significant levels of noise. Therefore, to further denoise the input, we employ the Non-local Means (NL-means) algorithm [8], which takes advantage of a specific property of natural images. That is, for a given small patch in a natural image, there tend to be other patches that are very similar. This is especially true for patches that are spatially close to each other [7]. The algorithm works by taking the average of similar patches and replacing the original values with this averaged result.

As per the description in [8], let \( v \) be a discrete image with noise. For a pixel \( i \), the denoised version of \( i \), calculated via the NL-means algorithm, is the weighted average of all the pixels within the image:

\[
NL(v)(i) = \sum_j w(i, j)v(j)
\]

where \( w(i, j) \) are the assigned weights determined by calculating the similarity between pixels \( i \) and \( j \) and must satisfy \( 0 \leq w(i, j) \leq 1 \) and \( \sum_j w(i, j) = 1 \). To calculate the similarity between \( i \) and \( j \), we first define a window, \( N_i \), around the pixel, which usually takes the form of a square but can be configured according to the task at hand. Then we use squared gaussian weighted Euclidean distance to quantify the similarity:

\[
E(i, j) = \|v(N_i) - v(N_j)\|^2_{L_2,a}
\]

where \( a \) denotes the standard deviation for the gaussian kernel. Then we can define the weights as:

\[
w(i, j) = \frac{1}{Z(i)}e^{-\frac{E(i, j)}{h^2}}
\]

where \( Z(i) \) is the normalizing factor formalized by:

\[
Z(i) = \sum_j e^{-\frac{E(i, j)}{h^2}}
\]

and \( h \) determines the decay of the exponential function and the decay of the weights as a function of the Euclidean distance.
4.2. **Non-local Means Denoising**

In the next chapter, we present the results from applying CLAHE and NL-means denoising on the test dataset (see Section 5.3).

![Image](image.png)

**Figure 4.4:** This depiction shows how CLAHE can increase contrast and NL-means denoising can help reduce noise after CLAHE has been applied.
Throughout this thesis, we have presented the training and validation results (summarized in Section 3.6) for various models, along with the intuition and theory behind their inception. In total, we have presented the baseline model that formed the benchmark and eight different models that we compared against the benchmark according to the performance achieved on the validation dataset. Model G (non-finetuned) and Model I (finetuned) achieved the best performance. The next step involves testing these models on the test dataset to decipher their performance in comparison to the benchmark. In a separate experiment, we preprocess images within the test dataset with color normalization and denoising techniques to determine whether results can be improved further for the best performing models.

In this chapter, Section 5.1 reports the results for the best models without finetuning; Section 5.2 reports the results for the best finetuned models; and in Section 5.3 we present the results from applying color normalization and denoising techniques to the test dataset.

### 5.1 Best Performing Models Without Finetuning

We used the test dataset to compare our best performing non-finetuned ESRS model (Model G) against the non-finetuned benchmark model (Model A). Table 5.1 and Figure 5.1 report the results. The validation performance of these models showed that Model G was the clear winner (see Section 3.6). On the test dataset, Model G again outperformed the benchmark. It improved on the benchmark’s top-1 rate by 8.2%, on the top-3 by 8.3% and on the top-5 by 6.1%. Table 5.2 and Table 5.3 present the confusion matrices for Model A and Model G respectively. We can clearly observe for both the models that two categories, Golf Club Head and Kayak, achieved disproportionately low performance. This poor performance lowers the overall accuracy of the models, which is why Table 5.1 also displays the accuracy rates with these two categories discounted. If these categories can be improved and brought to the level of performance achieved by all the other categories then we expect to garner much better accuracy rates in the future.
Table 5.1: Model G was the best performing non-finetuned model for the test dataset. Here, it is compared with the non-finetuned baseline model. We also show the results after excluding the Kayak and Golf Club Head categories to highlight that both models were poor at recognizing these two categories as the overall accuracy rates improve significantly if they are excluded.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model A</td>
<td>43.3%</td>
<td>65.8%</td>
<td>77.0%</td>
</tr>
<tr>
<td>Best ESRS - Model G</td>
<td>51.5%</td>
<td>74.1%</td>
<td>83.1%</td>
</tr>
</tbody>
</table>

Excluding Kayak Samples

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model A</td>
<td>45.2%</td>
<td>68.8%</td>
<td>80.4%</td>
</tr>
<tr>
<td>Best ESRS - Model G</td>
<td>54.0%</td>
<td>77.4%</td>
<td>86.5%</td>
</tr>
</tbody>
</table>

Excluding Golf Club Head Samples

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model A</td>
<td>45.1%</td>
<td>68.3%</td>
<td>79.4%</td>
</tr>
<tr>
<td>Best ESRS - Model G</td>
<td>53.7%</td>
<td>76.7%</td>
<td>85.6%</td>
</tr>
</tbody>
</table>

Excluding Kayak and Golf Club Head Samples

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model A</td>
<td>47.2%</td>
<td>71.6%</td>
<td>83.1%</td>
</tr>
<tr>
<td>Best ESRS - Model G</td>
<td>56.4%</td>
<td>80.3%</td>
<td>89.3%</td>
</tr>
</tbody>
</table>

Figure 5.1: Model G was the best performing non-finetuned model for the test dataset. Here, it is compared with the non-finetuned baseline model.

5.2 Best Performing Models With Finetuning

The baseline model and the best ESRS model were finetuned via a GoogLeNet model pretrained on ImageNet data (see Table 3.9). Table 5.4 and Figure 5.2 report the results. Both
Table 5.2: Confusion Matrix for Model A, the non-finetuned baseline.

<table>
<thead>
<tr>
<th>Truth</th>
<th>badminton</th>
<th>barbell</th>
<th>baseball</th>
<th>dumbbell</th>
<th>golfClub</th>
<th>kayak</th>
<th>roadBike</th>
<th>shoe</th>
<th>spinningBike</th>
<th>squash</th>
<th>swimming</th>
<th>tableTennis</th>
<th>tennis</th>
<th>treadmill</th>
</tr>
</thead>
<tbody>
<tr>
<td>badminton</td>
<td>52.17</td>
<td>2.90</td>
<td>0.00</td>
<td>4.35</td>
<td>2.90</td>
<td>0.00</td>
<td>7.97</td>
<td>5.80</td>
<td>2.17</td>
<td>1.45</td>
<td>0.00</td>
<td>1.45</td>
<td>15.22</td>
<td>3.62</td>
</tr>
<tr>
<td>barbell</td>
<td>2.05</td>
<td>33.56</td>
<td>4.11</td>
<td>32.39</td>
<td>1.37</td>
<td>0.00</td>
<td>3.42</td>
<td>0.00</td>
<td>6.85</td>
<td>0.68</td>
<td>1.37</td>
<td>7.53</td>
<td>1.37</td>
<td>5.48</td>
</tr>
<tr>
<td>baseball</td>
<td>2.01</td>
<td>2.01</td>
<td>60.40</td>
<td>8.72</td>
<td>0.00</td>
<td>0.67</td>
<td>0.00</td>
<td>12.75</td>
<td>0.67</td>
<td>0.00</td>
<td>1.34</td>
<td>4.70</td>
<td>5.37</td>
<td>1.34</td>
</tr>
<tr>
<td>dumbbell</td>
<td>0.52</td>
<td>12.95</td>
<td>5.18</td>
<td>40.93</td>
<td>3.11</td>
<td>3.11</td>
<td>1.04</td>
<td>2.59</td>
<td>2.59</td>
<td>0.52</td>
<td>7.25</td>
<td>8.29</td>
<td>9.33</td>
<td>2.59</td>
</tr>
<tr>
<td>golfClub</td>
<td>0.00</td>
<td>1.82</td>
<td>14.55</td>
<td>10.91</td>
<td>10.91</td>
<td>9.09</td>
<td>12.73</td>
<td>10.00</td>
<td>1.82</td>
<td>0.00</td>
<td>8.18</td>
<td>1.82</td>
<td>13.64</td>
<td>4.55</td>
</tr>
<tr>
<td>kayak</td>
<td>0.98</td>
<td>10.78</td>
<td>16.67</td>
<td>3.92</td>
<td>0.98</td>
<td>7.84</td>
<td>54.90</td>
<td>0.00</td>
<td>0.98</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.94</td>
<td></td>
</tr>
<tr>
<td>roadBike</td>
<td>0.00</td>
<td>0.52</td>
<td>2.59</td>
<td>0.52</td>
<td>0.52</td>
<td>0.00</td>
<td>88.08</td>
<td>2.59</td>
<td>1.04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.52</td>
<td>3.11</td>
<td>0.52</td>
</tr>
<tr>
<td>shoe</td>
<td>0.00</td>
<td>7.14</td>
<td>3.57</td>
<td>16.07</td>
<td>4.46</td>
<td>1.79</td>
<td>8.93</td>
<td>45.54</td>
<td>0.00</td>
<td>0.00</td>
<td>1.79</td>
<td>3.57</td>
<td>4.46</td>
<td>2.68</td>
</tr>
<tr>
<td>spinningBike</td>
<td>0.00</td>
<td>11.97</td>
<td>6.34</td>
<td>29.58</td>
<td>0.70</td>
<td>0.70</td>
<td>1.41</td>
<td>3.52</td>
<td>23.94</td>
<td>0.00</td>
<td>1.41</td>
<td>1.41</td>
<td>6.34</td>
<td>12.68</td>
</tr>
<tr>
<td>squash</td>
<td>4.96</td>
<td>0.71</td>
<td>9.22</td>
<td>4.26</td>
<td>2.13</td>
<td>0.00</td>
<td>13.48</td>
<td>7.09</td>
<td>2.84</td>
<td>5.67</td>
<td>0.71</td>
<td>1.42</td>
<td>46.10</td>
<td>1.42</td>
</tr>
<tr>
<td>swimming</td>
<td>0.00</td>
<td>0.62</td>
<td>6.25</td>
<td>13.12</td>
<td>1.88</td>
<td>0.00</td>
<td>10.62</td>
<td>5.00</td>
<td>7.50</td>
<td>0.00</td>
<td>38.75</td>
<td>1.25</td>
<td>2.50</td>
<td>2.50</td>
</tr>
<tr>
<td>tableTennis</td>
<td>3.23</td>
<td>4.03</td>
<td>4.03</td>
<td>4.84</td>
<td>12.90</td>
<td>0.00</td>
<td>0.00</td>
<td>2.42</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.55</td>
<td>4.03</td>
<td>0.00</td>
</tr>
<tr>
<td>tennis</td>
<td>16.37</td>
<td>0.44</td>
<td>3.54</td>
<td>2.65</td>
<td>0.44</td>
<td>0.00</td>
<td>3.98</td>
<td>3.10</td>
<td>9.29</td>
<td>1.33</td>
<td>0.00</td>
<td>0.88</td>
<td>54.42</td>
<td>3.54</td>
</tr>
<tr>
<td>treadmill</td>
<td>1.79</td>
<td>8.04</td>
<td>22.32</td>
<td>10.71</td>
<td>1.79</td>
<td>0.89</td>
<td>7.14</td>
<td>2.68</td>
<td>5.36</td>
<td>0.00</td>
<td>1.79</td>
<td>0.89</td>
<td>7.14</td>
<td>29.46</td>
</tr>
</tbody>
</table>

Table 5.3: Confusion Matrix for Model G, the best ESRS model (non-finetuned).

<table>
<thead>
<tr>
<th>Truth</th>
<th>badminton</th>
<th>barbell</th>
<th>baseball</th>
<th>dumbbell</th>
<th>golfClub</th>
<th>kayak</th>
<th>roadBike</th>
<th>shoe</th>
<th>spinningBike</th>
<th>squash</th>
<th>swimming</th>
<th>tableTennis</th>
<th>tennis</th>
<th>treadmill</th>
</tr>
</thead>
<tbody>
<tr>
<td>badminton</td>
<td>70.29</td>
<td>1.45</td>
<td>0.72</td>
<td>3.62</td>
<td>1.45</td>
<td>6.52</td>
<td>2.17</td>
<td>5.07</td>
<td>2.90</td>
<td>2.17</td>
<td>0.00</td>
<td>0.00</td>
<td>1.45</td>
<td>2.17</td>
</tr>
<tr>
<td>barbell</td>
<td>3.42</td>
<td>43.84</td>
<td>0.68</td>
<td>39.29</td>
<td>0.00</td>
<td>0.00</td>
<td>1.37</td>
<td>0.68</td>
<td>4.11</td>
<td>0.00</td>
<td>0.00</td>
<td>2.05</td>
<td>0.00</td>
<td>4.11</td>
</tr>
<tr>
<td>baseball</td>
<td>13.42</td>
<td>0.67</td>
<td>19.68</td>
<td>6.71</td>
<td>1.34</td>
<td>2.01</td>
<td>0.00</td>
<td>20.13</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4.70</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td>dumbbell</td>
<td>2.59</td>
<td>7.77</td>
<td>2.07</td>
<td>66.52</td>
<td>2.07</td>
<td>1.04</td>
<td>0.52</td>
<td>4.66</td>
<td>0.52</td>
<td>0.00</td>
<td>5.70</td>
<td>2.07</td>
<td>4.66</td>
<td></td>
</tr>
<tr>
<td>golfClub</td>
<td>7.27</td>
<td>0.91</td>
<td>9.09</td>
<td>5.45</td>
<td>13.64</td>
<td>9.09</td>
<td>15.45</td>
<td>9.09</td>
<td>0.91</td>
<td>0.00</td>
<td>0.00</td>
<td>12.73</td>
<td>15.45</td>
<td></td>
</tr>
<tr>
<td>kayak</td>
<td>0.98</td>
<td>3.92</td>
<td>1.96</td>
<td>1.96</td>
<td>10.78</td>
<td>3.92</td>
<td>50.80</td>
<td>2.94</td>
<td>2.94</td>
<td>0.00</td>
<td>0.00</td>
<td>7.84</td>
<td>2.94</td>
<td></td>
</tr>
<tr>
<td>roadBike</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>94.90</td>
<td>3.63</td>
<td>1.55</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>shoe</td>
<td>4.46</td>
<td>0.89</td>
<td>3.57</td>
<td>8.04</td>
<td>0.89</td>
<td>0.89</td>
<td>8.04</td>
<td>62.50</td>
<td>0.89</td>
<td>0.89</td>
<td>1.79</td>
<td>0.00</td>
<td>1.79</td>
<td>5.36</td>
</tr>
<tr>
<td>spinningBike</td>
<td>0.00</td>
<td>4.23</td>
<td>1.41</td>
<td>28.17</td>
<td>0.00</td>
<td>0.70</td>
<td>5.63</td>
<td>4.93</td>
<td>40.14</td>
<td>0.00</td>
<td>0.70</td>
<td>0.00</td>
<td>14.08</td>
<td></td>
</tr>
<tr>
<td>squash</td>
<td>7.80</td>
<td>0.00</td>
<td>2.84</td>
<td>1.42</td>
<td>0.71</td>
<td>1.42</td>
<td>15.60</td>
<td>0.71</td>
<td>0.00</td>
<td>30.50</td>
<td>1.42</td>
<td>0.00</td>
<td>32.62</td>
<td>4.96</td>
</tr>
<tr>
<td>swimming</td>
<td>1.25</td>
<td>0.62</td>
<td>12.50</td>
<td>6.88</td>
<td>1.25</td>
<td>0.62</td>
<td>10.00</td>
<td>4.38</td>
<td>3.12</td>
<td>0.62</td>
<td>52.50</td>
<td>0.00</td>
<td>0.00</td>
<td>6.25</td>
</tr>
<tr>
<td>tableTennis</td>
<td>10.48</td>
<td>0.81</td>
<td>0.00</td>
<td>4.84</td>
<td>12.10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>59.68</td>
<td>12.10</td>
<td>0.00</td>
</tr>
<tr>
<td>tennis</td>
<td>18.58</td>
<td>0.00</td>
<td>1.77</td>
<td>0.44</td>
<td>1.77</td>
<td>0.00</td>
<td>1.33</td>
<td>0.00</td>
<td>3.10</td>
<td>17.70</td>
<td>0.00</td>
<td>0.00</td>
<td>47.35</td>
<td>7.96</td>
</tr>
<tr>
<td>treadmill</td>
<td>0.89</td>
<td>1.79</td>
<td>5.36</td>
<td>14.29</td>
<td>1.79</td>
<td>1.79</td>
<td>12.50</td>
<td>3.57</td>
<td>4.46</td>
<td>0.00</td>
<td>1.79</td>
<td>0.89</td>
<td>0.89</td>
<td>50.00</td>
</tr>
</tbody>
</table>

Finetuned models clearly improve on the results achieved by their non-finetuned versions. Further, the finetuned ESRS model (Model I) clearly outperformed the finetuned baseline (Model H). It improved on the baseline’s top-1 rate by 4.5%, on the top-3 by 3.2% and on the top-5 by 2.8%. The best non-finetuned ESRS model improved on its benchmark by 6-8% but this finetuned version only achieved an improvement of 2-4%. Observe that during validation the finetuned baseline slightly outperformed the finetuned ESRS model but on the test dataset, the ESRS model actually achieved higher accuracy rates.

Table 5.5 and Table 5.6 present the confusion matrices for the baseline and ESRS models respectively. Again, the results show that both models struggled with the same two categories that troubled their non-finetuned versions: Golf Club Head and Kayak. Discounting both these categories leads to drastically improved performance for both models.
5.3. Results from Applying Color Normalization and Denoising

Table 5.4: Model I was the best performing finetuned model for the test dataset. Here, it is compared with the finetuned baseline model. We also show the results after excluding the Kayak and Golf Club Head categories to highlight that both models were poor at recognizing these two categories as the overall accuracy rates improve significantly if they are excluded.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model H</td>
<td>56.2%</td>
<td>79.0%</td>
<td>85.9%</td>
</tr>
<tr>
<td>Best ESRS - Model I</td>
<td>60.7%</td>
<td>82.2%</td>
<td>88.7%</td>
</tr>
</tbody>
</table>

Excluding Kayak Samples

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model H</td>
<td>60.7%</td>
<td>81.8%</td>
<td>88.0%</td>
</tr>
<tr>
<td>Best ESRS - Model I</td>
<td>63.5%</td>
<td>85.4%</td>
<td>91.2%</td>
</tr>
</tbody>
</table>

Excluding Golf Club Head Samples

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model H</td>
<td>58.8%</td>
<td>82.0%</td>
<td>88.4%</td>
</tr>
<tr>
<td>Best ESRS - Model I</td>
<td>63.9%</td>
<td>85.9%</td>
<td>92.1%</td>
</tr>
</tbody>
</table>

Excluding Kayak and Golf Club Head Samples

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - Model H</td>
<td>61.6%</td>
<td>85.1%</td>
<td>90.8%</td>
</tr>
<tr>
<td>Best ESRS - Model I</td>
<td>67.0%</td>
<td>89.5%</td>
<td>94.9%</td>
</tr>
</tbody>
</table>

Figure 5.2: Model I was the best performing finetuned model for the test dataset. Here, it is compared with the finetuned baseline model.

5.3 Results from Applying Color Normalization and Denoising

We preprocessed test images with CLAHE and NL-means denoising to see whether we would see an improvement in predictive accuracy (see Figure 4.1). Since our ultimate goal is to
produce the best possible accuracy rate, we decided to use the best ESRS non-finetuned and finetuned models along with their respective benchmarks during these experiments. CLAHE was applied independently first and then CLAHE and NL-means were both applied to see how the results differed. The NL-means denoising algorithm is used in combination with CLAHE because the color normalization technique can increase the level of noise within an image. Thus, to reduce the noise, we decided to combine it with a denoising method. Table 5.7, Figure 5.3 (non-finetuned) and Figure 5.4 (finetuned) report the results. All results in this section take the entire test dataset into account.
5.3. Results from Applying Color Normalization and Denoising

Table 5.7: The top-1, top-3 and top-5 accuracies achieved by the best ESRS models on the test dataset after preprocessing with CLAHE and NL-means denoising.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-finetuned Models</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline - Model A</td>
<td>43.3%</td>
<td>65.8%</td>
<td>77.0%</td>
</tr>
<tr>
<td>Baseline - Model A (CLAHE)</td>
<td>42.2%</td>
<td>63.1%</td>
<td>75.7%</td>
</tr>
<tr>
<td>Baseline - Model A (CLAHE + NL-means)</td>
<td>43.8%</td>
<td>65.7%</td>
<td>76.8%</td>
</tr>
<tr>
<td>Best ESRS - Model G</td>
<td>51.5%</td>
<td>74.1%</td>
<td>83.1%</td>
</tr>
<tr>
<td>Best ESRS - Model G (CLAHE)</td>
<td>51.9%</td>
<td>73.0%</td>
<td>81.9%</td>
</tr>
<tr>
<td>Best ESRS - Model G (CLAHE + NL-means)</td>
<td>52.1%</td>
<td>75.1%</td>
<td>82.9%</td>
</tr>
<tr>
<td><strong>Finetuned Models</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline - Model H</td>
<td>56.2%</td>
<td>79.0%</td>
<td>85.9%</td>
</tr>
<tr>
<td>Baseline - Model H (CLAHE)</td>
<td>63.6%</td>
<td>82.4%</td>
<td>87.7%</td>
</tr>
<tr>
<td>Baseline - Model H (CLAHE + NL-means)</td>
<td>64.8%</td>
<td>82.8%</td>
<td>88.5%</td>
</tr>
<tr>
<td>Best ESRS - Model I</td>
<td>60.7%</td>
<td>82.2%</td>
<td>88.7%</td>
</tr>
<tr>
<td>Best ESRS - Model I (CLAHE)</td>
<td>65.6%</td>
<td>81.7%</td>
<td>87.5%</td>
</tr>
<tr>
<td>Best ESRS - Model I (CLAHE + NL-means)</td>
<td>67.0%</td>
<td>83.3%</td>
<td>88.3%</td>
</tr>
</tbody>
</table>

5.3.1 Results from Applying CLAHE

The results for preprocessing the input data with CLAHE are mixed. For the non-finetuned models, the baseline performed better in all three metrics (see Figure 5.3) when CLAHE was not applied. The best ESRS model (Model G) experienced a slight increase in top-1 accuracy when CLAHE was applied but the top-3 and top-5 accuracy rates deteriorated. Thus, overall for the non-finetuned models, preprocessing with CLAHE led to a decrease in predictive accuracy.

The finetuned models, on the other hand, responded much better to CLAHE preprocessing (see Figure 5.4). Almost all metrics for both the baseline and the best ESRS model (Model I) saw a significant improvement. The only blemishes were the top-3 and top-5 metrics for Model I, which deteriorated slightly when CLAHE was applied. Thus, overall for the finetuned models, CLAHE preprocessing had a mostly positive impact on predictive accuracy.

5.3.2 Results from Applying CLAHE and NL-means Denoising

The results from preprocessing the input data with CLAHE and NL-means denoising are mostly positive. For the non-finetuned models (see Figure 5.3), applying these techniques to the baseline model led to a clear improvement in all metrics over applying CLAHE only. However, in comparison to the original model (no preprocessing), only the baseline’s top-1 metric performed slightly better while the top-3 and top-5 metrics were slightly worse. In the case of the best ESRS model (Model G), applying CLAHE and denoising improved the metrics across the board when compared to only applying CLAHE. When compared to the original
5.4 Addressing Challenges

In Section 1.4, we outlined the challenges that needed to be overcome in order to produce a quality model capable of generalization. We have shown that Challenge I has been overcome as we have created models that have reduced overfitting and produced good results against the training dataset. The validation results from Model F demonstrate that overfitting was significantly curtailed and the test results from Model I returned a strong top-5 accuracy of 88.7%.

Challenge II addressed the problem of small inter-category variations. Table 5.3 and Table 5.6 show the confusion matrices of the best non-finetuned and finetuned ESRS models respectively. The results do indicate that the three racquet categories we expected to cause problems (see Figure 1.5) are being confused with one another. The Squash Racquet category caused the
5.4. Addressing Challenges

Figure 5.4: This chart depicts the impact that CLAHE and NL-means denoising had on the top-1, top-3 and top-5 accuracies of the best finetuned ESRS model (Model I) and the finetuned baseline (Model H).

most confusion, mostly with the Tennis Racquet category. Again, this makes sense as visually squash racquets can closely resemble tennis racquets, especially from side-on angles. The non-finetuned model correctly labeled 30.5% of Squash Racquet samples with 32.6% mislabeled as tennis racquets and 7.8% mislabeled as badminton racquets. The finetuned model, however, performed very poorly. It mislabeled 76.6% of samples as tennis racquets. Higher accuracy rates were achieved for the Badminton Racquet category: 70.3% and 65.9% were correctly labeled for non-finetuned and finetuned respectively. Lastly, for the Tennis Racquet category the finetuned model performed well by correctly labeling 76.1% of the samples but the non-finetuned model only managed to correctly label 47.4% of the samples. Overall, the results for this challenge were mixed. More training data containing images taken from every possible angle and orientation are required to improve performance.

Overall, the Golf Club Head and Kayak categories were the most difficult to predict. Removing these two categories from the equation greatly boosted the overall accuracy rates achieved by each model (see Tables 5.1 and 5.4). This is primarily due to the variance in the training and testing datasets (Challenge III). This is apparent as there is a large gap between the overall top-1 and top-5 accuracy rates for the validation dataset versus the test dataset. Predictive performance is much better on the validation dataset. For these two categories, the trained models were not able to overcome the deficiencies in each dataset. In the case of the Kayak category, the test dataset was deficient as it did not contain any bodies of water. Alternatively, for the Golf Club Head category, the training dataset seemed to be lacking the types of images required to achieve high accuracy rates with the test data.
Challenge II’s large intra-category variation problem along with Challenges IV and V were addressed adequately as, despite these challenges, we were able to produce better results on the test dataset in comparison to the benchmark models: top-1 of 60.7% and top-5 of 88.7% (see Figure 5.2). In the future, results can be boosted further by improving the quality and increasing the size of the training dataset. Figure 5.6 provides examples from the test dataset that were particularly difficult to recognize.

Challenge VI highlights the difficulty that low contrast images pose and their abundance when working with mobile phones. We showed that color normalization and denoising techniques like CLAHE and NL-means can improve overall classifier performance (see Table 5.7). Also, we show that many low contrast images that were misclassified by the original classifier were correctly classified by the same model if preprocessed with CLAHE. Figure 5.5 provides some examples. This is not the case for every low contrast image but we have demonstrated that overall accuracy rates can be boosted by preprocessing with CLAHE and NL-means denoising.
Figure 5.5: Low contrast images from the test dataset that were recognized correctly due to CLAHE preprocessing. For each image set, the left image is the original and the right image is the result of applying CLAHE. The correct label is printed under each image set and the assigned probability for the correct label is highlighted by the red bar, if it is present within the top-5 results.
Figure 5.6: Various images from the test dataset that contained significant noise (e.g., humans, background clutter, multiple categories). The correct label is printed under each image and the assigned probability for the correct label is highlighted by the red bar, if it is present within the top-5 results. See Appendix A for more examples.
Chapter 6

Conclusions and Discussion

In this chapter, we conclude this thesis by summarizing our contributions (Section 6.1) and discussing future work (Section 6.2).

6.1 Conclusions

Today’s mobile health management systems rely on manual input or sensing tools such as GPS and accelerometers to record user data. The onboard cameras ubiquitously packaged with today’s phones remain underused. Manual input is too tedious, which is why an opportunity exists for the camera to become a practical and fun alternative for recording user input. This is why, in this thesis, we propose an Exercise and Sports Equipment Recognition System (ESRS) capable of recognizing 14 different equipment categories from raw image data. Our system can easily be incorporated with mobile platforms to create a mobile health framework that can detect equipment via an image and map it to a physical activity to deduce metrics such as calories burnt per minute.

We created a variety of predictive models by training deep convolutional neural networks on a relatively small sized dataset consisting of about 16 thousand training and 4 thousand validation samples. Along with proposing and creating ESRS, a system capable of recognizing 14 categories, our main contributions also include improving on the state-of-the-art GoogLeNet architecture for this dataset. Our best performing model achieved a top-1 accuracy of 60.7%, a top-3 of 82.2% and a top-5 of 88.7% on the test dataset. We also demonstrate how overfitting occurs when training a deep network with small amounts of training data and show how this problem can be reduced by employing L2 regularization and increasing dropout in the final fully connected layer. Furthermore, we propose a preprocessing scheme that uses Contrast Limited Adaptive Histogram Equalization (CLAHE), a color normalization technique, and Non-local Means denoising to improve accuracy on low contrast images and boost overall prediction accuracy. Combining these techniques with our best performing model leads to even better accuracy rates: top-1 accuracy of 67.0% and a top-3 of 83.3%. However, not all results are positive. The top-5 accuracy of 88.3% is slightly lower than the accuracy rate achieved by the original model.
6.2 Future Work

The accuracy rates achieved by ESRS are promising but further improvement is very achievable by increasing the overall training dataset size and improving the quality of training samples for two categories (Golf Club Head and Kayak) that showed disproportionately poor results.

We would also like to increase the number of categories that ESRS is capable of recognizing. Common types of equipment such as soccer ball, yoga mat, baseball bat etc. can be added to the system. Furthermore, all the categories we have added thus far cover activities that physically active individuals find engaging. The next step would be to explore how this system can cover more basic activities like walking or even standing to ensure that the system can also be used by individuals that only engage in limited physically activity.

In this work, CLAHE and NL-means denoising were used to preprocess test images fed to the predictive models. We would also like to explore how predictive accuracy would be impacted if these preprocessing techniques were applied to the training dataset. This approach may lead to improvements and reduce the time required to recognize a test image as the preprocessing stage during testing would be removed.

Lastly, crawled images from the Internet can contain annotations and animations (see Section 1.4, Challenge III) that are usually not present in images extracted directly from mobile phones. Mobile phones also tend to produce a disproportionate amount of low contrast images (see Section 1.4, Challenge VI). Thus, a disparity exists between crawled images and mobile phone images. This work used crawled images for training and mobile images for testing. In the future, automated techniques can be explored to reduce this difference between the two types of images and further boost predictive accuracy. A preliminary approach we would like to explore involves a histogram transformation scheme that would create an average intensity histogram describing the set of testing images and modify the set of training images so that the average histogram produced is a closer match to the testing set’s histogram.
Bibliography


BIBLIOGRAPHY


Appendix A

More Recognition Results on Test Images from Model I

Figure A.1: Recognition results on test images. The correct label is printed under each image and the assigned probability for the correct label is highlighted by the red bar, if it is present within the top-5 results.
Figure A.2: Recognition results on test images.
Figure A.3: Recognition results on test images.
| Figure A.4: Recognition results on test images. |
Figure A.5: Recognition results on test images.
Figure A.6: Recognition results on test images.
Curriculum Vitae

Name: Siddarth Kalra

Post-Secondary Education and Degrees:
Seneca College of Applied Arts and Technology
Toronto, ON
2005 - 2009 Bachelor of Applied Technology, Software Development

University of Western Ontario
London, ON
2014 - 2016 M.Sc. Computer Science

Related Work Experience:
Teaching Assistant and Research Assistant
The University of Western Ontario
2014 - 2015