Glenforest Secondary School

# To What Extent can Histogram Equalization Improve the Accuracy of a CNN Image Recognition AI for Low Contrast Images?

Ethan Chan

Extended Essay

Pauline Quan

November 30, 2021

# 1. Introduction

The contrast of an image is often a significant factor in regards to the level of detail that can be perceived by a viewer. In a situation where the dynamic range of an image is extremely narrow, such as in old photographs, finer features may be hard to distinguish, lacking the contrast to differentiate themselves from the rest of the image (Krukar, 2021). To alleviate this issue, contrast enhancement techniques, such as histogram equalization (HE), modify the intensities (brightnesses) of pixels in an image to widen the dynamic range (Asamoah et al., 2018), improving the contrast between details in the image to allow for greater visual clarity (Figure 1).

Figure 1: An image processed by HE (right) compared to the original image (Capper, 2006)

Such contrast enhancement techniques can be used for many applications in the real world, such as on x-ray scans, ultrasounds, and the restoration of old photos, allowing human viewers to make out details that previously may not have been as clear (Asamoah et al., 2018). However, in this day and age, humans are not the only ones capable of “seeing”. Computer vision is a rapidly growing subfield of artificial intelligence whose goal is to emulate human vision. One particular task in computer vision is the classification of the contents of an image, where a machine attempts to label the object(s) present in the image (Brownlee, 2019). This paper seeks to investigate to what extent (if any) does histogram equalization affect the accuracy of an image classification AI.

## 2. Theory

### *i. Image Histograms*

Images are arrangements of pixels, each possessing their own intensities, representing how bright or dark that pixel is. If the image is coloured, pixels can contain intensities for each RGB (red, green, blue) channel, in order to form a certain colour. However, grayscale images only possess a singular intensity value. Intensity values range from 0 (black) to a maximum value (white) which is dependent on how many bits are used to store each pixel. Generally, 8 bits are allotted per pixel, meaning there are $2^8$, or 256 possible values the intensity may take. Due to the fact that 0 is a possible value, the maximum intensity for an 8-bit pixel is 255 (NPS, 2008).

Following this definition of an image, it is possible to turn the intensity values comprising an image into a histogram (referred to as an intensity histogram), where each value on the x-axis would represent the number of pixels with that intensity in the corresponding image. For example, an 8-bit grayscale image would have 256 values on the x-axis, for each possible intensity (see Figure 2).

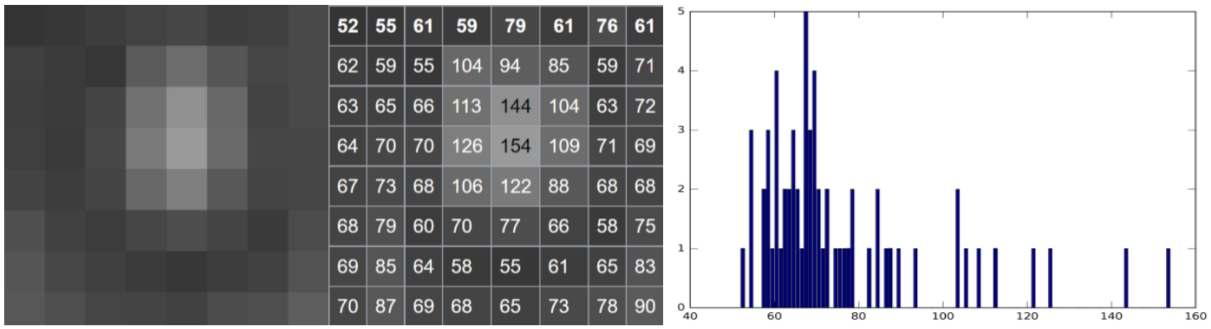



Figure 2: An 8x8 8-bit grayscale image (Cburnett, 2008), the intensity of each pixel (Cburnett, 2008), and the resulting intensity histogram (full x-axis is not shown, as there are no values past 40 or 160) (Rasulnrasul, 2005)

Darker images will have the peaks of their histogram around the left side (closer to 0/black), while brighter images will have peaks around the right side (white). The more contrast in a picture, the more spread out and even the peaks of the histogram will be, as there will not be an excessive amount

of a single brightness (which reduces contrast) (Fisher et al., 2003). One extension of the intensity histogram is the cumulative intensity histogram. Instead of each x-value representing the number of pixels with that intensity, each x-value represents the number of pixels having an intensity less than and up to that x-value (Easwaran, 2009). For example, x = 64 would contain the number of pixels with intensities ranging from [0, 64].

Intensity histograms can be represented as:

$$H = \{ h(k) \mid k = 0, 1, 2, \ldots, L\text{-}1 \}$$

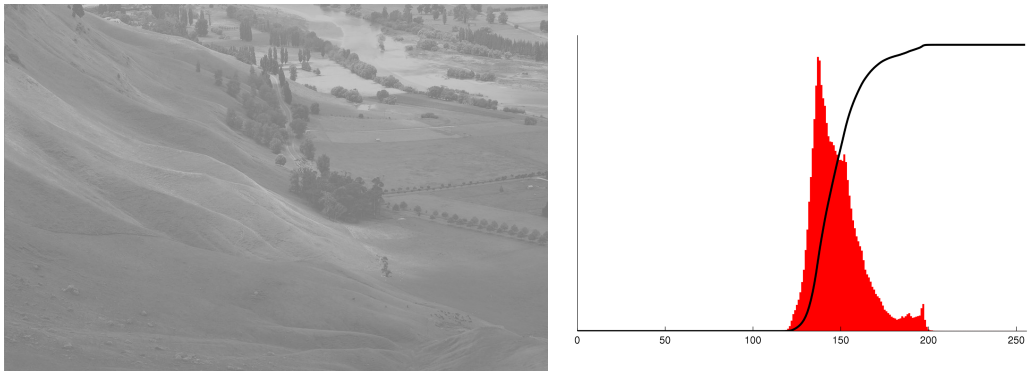
In the histogram $H$, the value $h(k)$ represents the number of pixels with intensity $k$, which ranges from 0 to the number of possible intensity levels $L$, minus 1. Following this, the cumulative intensity histogram can be calculated (Easwaran, 2009):

$$c(k) = \sum_{i=0}^{k} h(i)$$

Summing the histogram values from 0 up to an intensity value $k$, giving the number of pixels with intensities in the range [0, $k$].

*ii. Histogram Equalization*

The intent of histogram equalization is to create a uniform/equalized histogram, where the frequency of each intensity is (ideally) the same. An image with such a histogram would possess greater contrast and dynamic range, having been stretched out to the full intensity range (see Figure 3.).



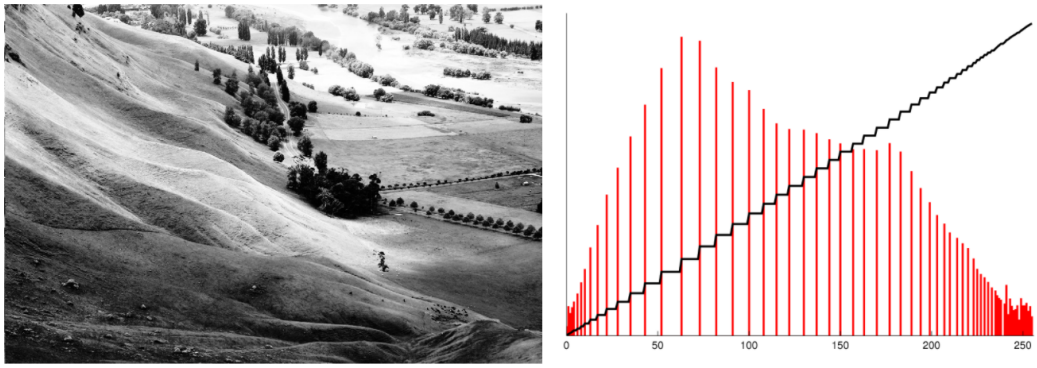



Figure 3: The histograms (Jarekt, 2008) of the original image (Capper, 2006) vs. the histogram equalized image. The red bars represent the histogram, while the black line represents the cumulative distribution function (CDF).

However, it is unrealistic for the equalized histogram to be completely flat (uniform in frequency), as that would mean that the taller (more frequent) intensities would need to be cut down, and vice versa for the shorter (less frequent) intensities. This would adversely affect the information contained in the image. For example, in Figure 3, it is clear that there are many dark pixels (the tall bars), which form the lower half of the image. If some of those pixels were to be converted into lighter intensities, the information conveyed by the equalized image would be inaccurate to the original image (Fisher et al., 2003). To circumvent this, the HE process aims instead to linearize the CDF.

The CDF is a function where CDF($k$) represents the probability of a pixel in the image being of intensity $k$ or lower (Deisenroth et al., 2020). Thus, a linear CDF would represent even frequencies for all intensities, because if all intensities have roughly the same frequency, as the x-value increases, CDF($k$) would increase at a constant rate, resulting in a linear pattern. This linearization can be accomplished by arranging the histogram bars such that high frequency (tall) bars are spread out, while low frequency (short) bars are clustered together, resulting in an even distribution overall of pixels. HE does just that, and can be thus considered as a type of intensity transformation, which is a process where all pixels of a certain intensity in an image are remapped to a new intensity (Easwaran, 2009). This type of operation

can be performed using a lookup table (LUT), which is a table containing all possible input intensity values, and the corresponding remapped output values. By inputting all pixels and remapping them according to the LUT, a whole image can be transformed (Easwaran, 2009).

The derivation of the LUT is the most crucial point of the HE process. To do this, it should be understood how to calculate what any given intensity in an image should be remapped to.

Firstly, in an ideal equalized histogram, the number of pixels for any intensity should be identical, and thus equal to $\frac{N}{L}$, $N$ representing total number of pixels, $L$ representing total intensity levels (Easwaran, 2009). Let $r_k$ represent the original intensity, and $s_k$ the remapped intensity. To maintain informational integrity, the number of pixels with intensities up to $r_k$ should be equivalent to the number of pixels up to $s_k$. Combined together gives the equation (Easwaran, 2009):

$$\sum_{i=0}^{r_k} h(i) = \left(\frac{N}{L}\right)(s_k + 1)$$

Factoring out $\frac{N}{L}$ and rearranging:

$$s_k = \left(\frac{L}{N}\right)\sum_{i=0}^{r_k} h(i) - 1$$

Value $s_k$ must be an integer, as intensity values cannot be decimals. It is likely that the right side of the equation will evaluate into a decimal, thus the rounded value will be taken instead. It is also possible for $s_k$ to evaluate into a negative, which is not possible for an intensity value. In such a case, $s_k$ is set to 0 (Easwaran, 2009). Understanding how to transform any intensity into a remapped intensity, the LUT can be easily generated and applied to the image, ultimately resulting in a histogram equalized image.

### *iii. Convolutional Neural Networks (CNNs) and Image Classification*

The image recognition AI used in this investigation will be a CNN, which is the most common neural network used to classify images (Peregud, 2020). CNNs first look for patterns and features in the image, through the usage of “filters”. Each filter takes a size specified by the user (commonly 3x3 or 5x5),

and is moved across and down the image, returning a value calculated using a convolution operation for each point in the image (Figure 4) (Stewart, 2016).

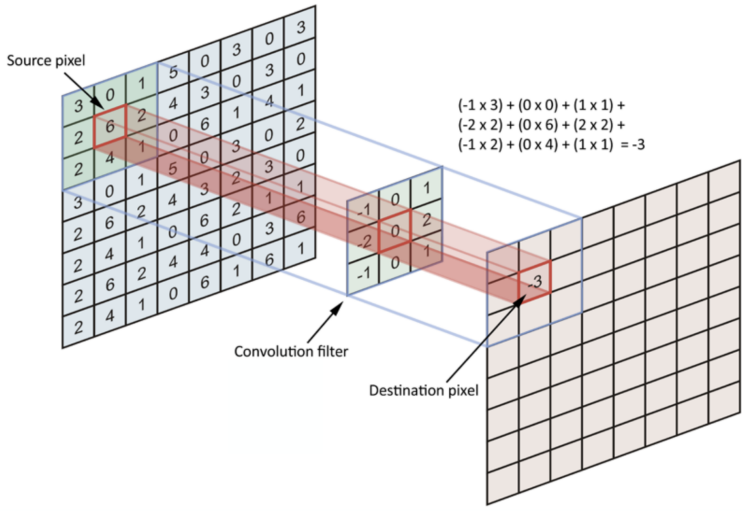



Figure 4: A filter performing a convolution operation and returning a value (Stewart, 2016)

Filters can look for many different patterns. For instance, a filter could look for the edges of objects in an image (see Figure 5).

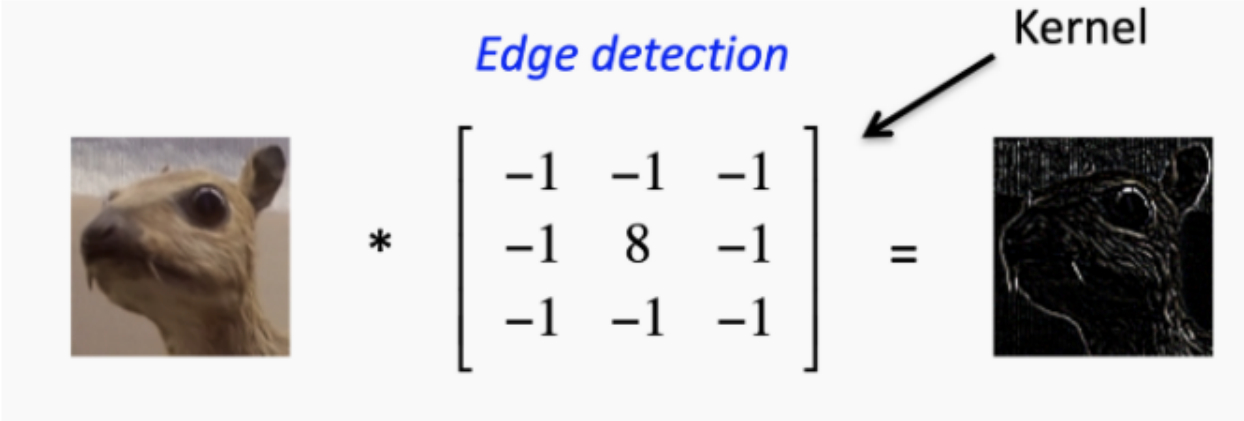



Figure 5: A filter designed to detect the edges in an image (Stewart, 2016)

Edges in an image normally contrast with their surrounding pixels, so passing this filter over an edge, the -1's would be multiplied by fairly low intensities, while the 8 would be multiplied by a higher intensity, resulting in a high returned value (which can be seen in the white in Figure 5). On the other hand, if the filter passes over a uniform area (with no edge), all the intensities would cancel out once multiplied by the filter ( $8 \times k + 8 \times (-1) \times k = 0$), resulting in the black areas in the product image.

Once a filter has passed every point, it will have returned a matrix of values known as a feature map, or activation map (Stewart, 2016). Each CNN possesses multiple filters, and thus there will be equivalently many feature maps, which are necessary in order for the network to successfully identify patterns. This process of convolution is the first layer of a CNN (Stewart, 2016).

The next layer in the CNN downsamples all the activation maps into smaller matrices to save on storage and computing power. The CNN will look at patches of an activation map (e.g. a 2x2 area), and take the largest value from that area, putting it into a new, smaller matrix, which is a process known as max pooling (Figure 6). Only the location of the largest values, representing the strongest correlation to each feature, remain, while lesser values are discarded (Nicholson, 2020).

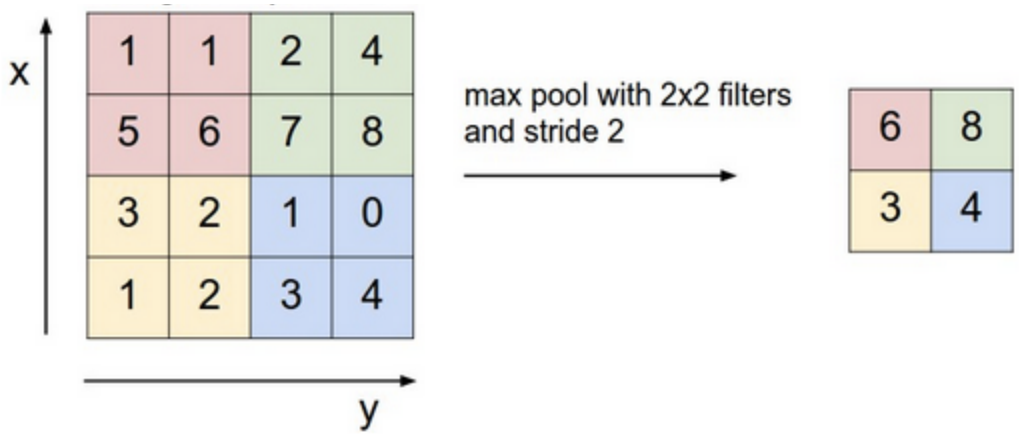



Figure 6: The downsampling and max pooling process (Nicholson, 2020)

Typical CNNs will have additional convolution layers after the first downsampling layer, further processing the data. Although the example of the edge filter was fairly clear to humans, the patterns processed by CNNs are rarely that intuitive, and become more and more abstract as more layers of convolution and downsampling are performed (Nicholson, 2020).

Once a CNN has ran all convolution and downsampling layers, the remaining activation maps pass through a neural network to finally output the image classification. Neural networks (NNs) are built using an arrangement of artificial neurons, separated into input, hidden, and output layers (Figure 7) (Sanderson, 2017).

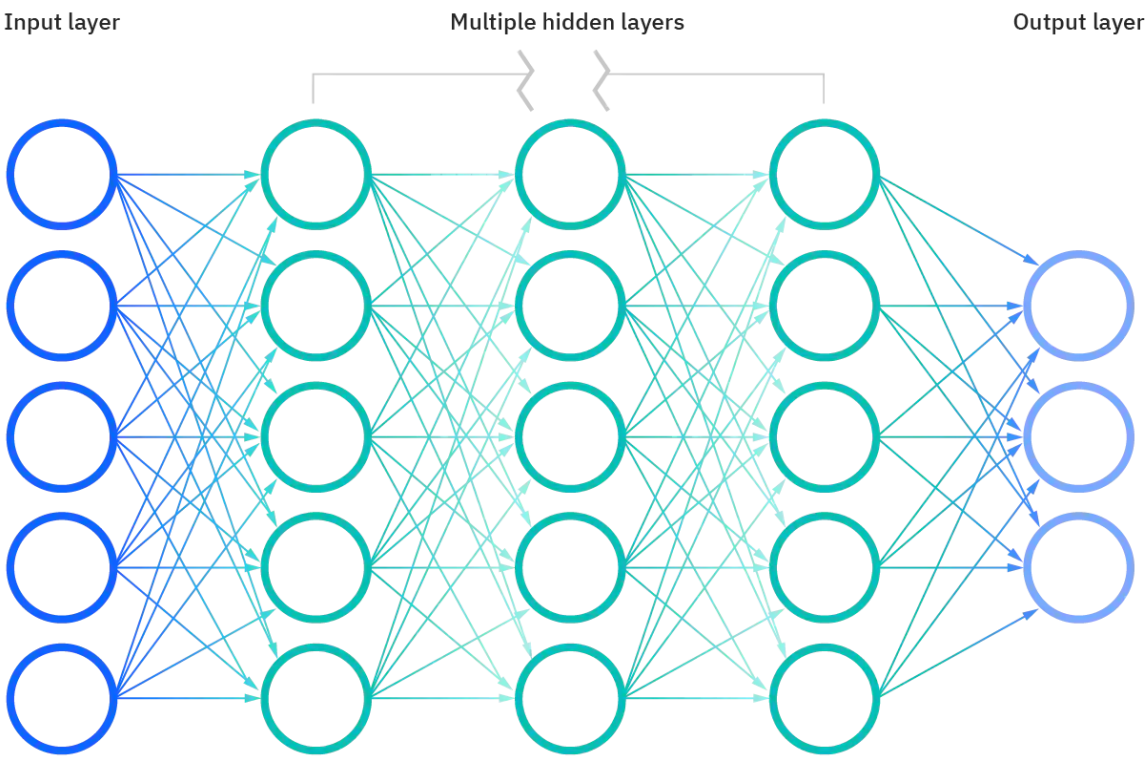



Figure 7: A traditional neural network with input, hidden, and output layers (IBM, 2020)

Every neuron in a NN connects to all neurons in both the previous and next layer. Each neuron will hold a value generally in the range [0, 1], known as an activation, either from calculating it given inputs from previous neurons, or from being given an input (activation maps), while each connection between neurons will hold a multiplier known as a weight (Sanderson, 2017). Neurons pass their activations through all their connections, which are then multiplied by the weight of each connection, and given as input to the next neuron . Once a neuron receives all inputs from the neurons connected to it, it runs a function known as the activation function, which sums up all the values it received, adds an arbitrary value known as a bias, and then converts it back into a value in the range [0, 1], eventually reaching the output layer (Sanderson, 2017). Each neuron in the output layer represents one possible image classification/label (e.g. car, bird, street), and its activation represents the probability that the input image is part of its corresponding classification. Values closer to 1 represent the neuron is confident in the image being of that class, while values closer to 0 represent the opposite (Sanderson, 2017). By using

a labelled dataset where input images have corresponding answers, the network can compare its output to the actual response, and calculate how far it was from getting the right answer, known as the cost (higher cost is farther away) (Sanderson, 2017). The network wishes to minimize the cost, and will slowly adjust the weights and biases present in the network, as more and more training data is fed to the NN, ultimately resulting in more accurate output (Sanderson, 2017).

## 3. Setup

The intent of this paper is to investigate the extent to which histogram equalization is able to affect the accuracy of an image classification CNN. Thus, to perform this investigation, a couple components were needed.

### *i. Image Dataset*

A large set of testing images are required to accurately gauge the accuracy of the CNN. For this experiment, Google's Open Images Dataset was used to source 1000 images, along with their labels, to test the CNN with. However, a few modifications had to be made before they could be used. The images were first converted to a grayscale colour space using Python's OpenCV (Open Source Computer Vision) library, so that the HE process would only need to be applied once. Next, the contrast of the images were reduced using Python's PIL (Python Image Library) library, so that the effects of HE on the images could be more clearly seen. Finally, the HE process was applied to the lower contrast images, ultimately resulting in 3 sets of 1000 images to be tested: the grayscale original images, the lowered contrast images, and the histogram equalized images (see Figure 8).

Figure 8: From left to right, a low contrast image, the histogram equalized image, and the original grayscale image (all sourced from Google's Open Images Dataset)

Each original grayscale image possessed its own ID, which was shared with the lowered contrast and histogram equalized variants to identify which classifications they contained. The JSON file containing all 1000 IDs and corresponding labels was sourced from Google.

### *ii. Histogram Equalization Implementation*

Although a naive implementation of the histogram equalization process was written (Appendix A), it was found to be too slow. However, Python's OpenCV (Open Source Computer Vision) library contains an optimized histogram equalization function that was used instead in this investigation to transform the lowered contrast imageset.

### *iii. Convolutional Neural Network*

The CNN used to evaluate the images was Imagga's online image recognition API. The API, once accessed, would return a JSON composed of all the labels it thought fit the image. An online CNN was used, firstly due to the fact that results gained from this investigation would be far more replicable for others, and secondly due to storage limitations restricting the training of a new CNN on the PC used in this experiment. However, the code used to bring all the components (the various libraries mentioned above, and usage of this API) together was written by myself.

## 4. Experimentation

### *i. Independent Variable*

The independent variable of this investigation was the usage of the histogram equalization process to enhance the image, before classification by the CNN. One imageset contained the low contrast images, whilst another contained the histogram equalized low contrast images, with the last containing the

original grayscale images. The original grayscale images are intended to provide a baseline of how the CNN would perform on “normal” images.

### *ii. Dependent Variables*

1. Number of Correct Labellings

If a labelling returned by the CNN was part of the official labels provided by Google, then it was considered as correct. The CNN was restricted to outputting its top 5 most confident labels, to ensure that any correct labelling was not due to the sheer number of tags returned.

2. Average Confidence

The confidence returned by the CNN is an indication of how sure the CNN is in its returned results, which will provide another perspective in analysing how HE affects CNN accuracy. The average confidence was calculated by summing all the confidences of the correct labellings, then dividing by the final number of correct classifications.

### *iii. Constants*

1. The CNN used - Imagga’s Image Recognition API
2. Number of images and image contents - 1000 images were selected from Google’s Open Images Dataset, and were transformed into the 3 imagesets used in testing
3. Image Labels - All classifications returned by the CNN were checked against the same list of labels, which were sourced from Google’s Open Images Dataset

### *iv. Procedure*

1. One Python dictionary for each imageset was initialized, with all image IDs entered as keys
2. The CNN was fed the images from all the imagesets

3. The returned classifications were put into their respective dictionaries corresponding to their image ID
4. Each dictionary's image labels were checked against the official labellings, and correct labellings were tallied
5. The confidences of the correct labellings were summed up for each imageset
6. The average confidence for each imageset was calculated by dividing the sum by the total number of correct labellings for that imageset
7. The final results were outputted into a JSON file where each imageset corresponded to the number of correct labellings the CNN made with it.

   The programs used to run these tasks can be found in Appendix B.

## 5. Results

### *i. Tabulated Data*

The following table contains the number of correct labellings each imageset produced when fed to the CNN, as well as each imageset's average confidence.

| | Grayscale | Histogram Equalized | Low Contrast |
|---|---|---|---|
| Correct Labellings | 1042 | 890 | 656 |
| Average Confidence | 71.181 | 68.071 | 59.786 |

Table 1: Number of correct labellings and average confidence for each imageset

### *ii. Processed Data*

To better understand the results, calculations were performed on the raw data presented in Table 1.

1. Accuracy Table

The accuracy of the CNN's guesses for each imageset was calculated by dividing the correct labellings returned by the total number of labels (9134).

| Grayscale | Histogram Equalized | Low Contrast |
|---|---|---|
| 11.408% | 9.744% | 7.182% |

Table 2: CNN's guess accuracy for each imageset

2. Percentage Change in Correct Labels

The percentage difference between the number of correct labellings for each imageset was calculated. Each value represents how much more/less the imageset of the row had vs. the imageset of the column (e.g. the grayscale imageset had as many correct labellings as the histogram equalized imageset + 17.079% extra).

| | Grayscale | HIstogram Equalized | Low Contrast |
|---|---|---|---|
| Grayscale | 0.000% | +17.079% | +58.841% |
| HIstogram Equalized | -14.587% | 0.000% | +35.671% |
| Low Contrast | -37.044% | -26.292% | 0.000% |

Table 3: Percentage difference in number of correct labels

## 6. Analysis

### *i. Observations*

Clearly, the original grayscale images performed the best, having roughly 17% more correct labels than the histogram equalized set, which in turn had 35.671% more correct labels than the worst-performing low contrast set. The average confidence and accuracy of each set followed the same trend, with grayscale images having the highest average confidence and accuracy, followed by the histogram equalized set, with the low contrast set performing the worst. However, in terms of CNN image recognition overall, the accuracy of all imagesets can be considered rather terrible. If optimized properly, CNN's can have accuracies of 90%+ (Rizvi, 2020), a far cry from the meagre 11.4% displayed by the grayscale imageset, the highest accuracy in this experiment.

*ii. Understanding the Results*

1. Why was the HE set able to outperform the low contrast set?

The main feature of CNNs is the fact that they employ filters that operate directly on pixel intensity values to try and detect certain features present in the image to perform image classifications (Nicholson, 2020). For instance, as mentioned earlier, a filter could look for edges in an image, returning a high value when an edge is present, and a low one otherwise. However, the basis for the edge filter to be able to recognize the edge is the fact that there is enough difference in the pixel intensities for it to generate a high value (Nicholson, 2020). Thus, even if a low contrast image possessed an edge, the difference in pixel intensities may not be enough for the filter to recognize it as such. This scenario could occur for other filters as well, and would interfere with the foundation upon which CNNs recognize images. The HE imageset would not face such an issue with its enhanced contrast, explaining its improved performance.

2. Why did the original grayscale images outperform the HE set?

The assumption that histogram equalization improves an image hinges on the idea that pixel intensities should be spread across the entirety of the intensity range for an improved image (Easwaran, 2009). However, this may not be true. When performing HE, pixel intensities with high frequencies are spread across large gray intervals, leaving smaller details to disappear between objects containing those high-frequency intensities (Asamoah et al., 2018), negatively impacting the feature recognition process of CNNs. Additionally, HE is a process impartial to the input image. Regardless of the contents of the image, everything will be enhanced, even imperfections like noise. Papers have shown noise to be heavy detriments to the performance of CNNs (Dodge & Karam, 2016), and with the contribution of these two factors, it can be seen why the HE images did not perform as well as the grayscale images.

## 7. Conclusion

Looking at the data collected in this investigation, it can be concluded that depending on the usage scenario, histogram equalization can provide a simple means to improve the accuracy of an image classification CNN. The usage of HE will only be effective should the images being used possess low contrast. Although there are many means of improving the accuracy of a CNN, most of them are much more involved, requiring careful tweaking of parameters, or a large scale increase in training data (Pawar, 2018). Thus, HE can provide a low-cost alternative to such methods, given the right scenario. Real-world applications of these results could include processing old images, which generally have low contrast due to technological limitations of the time.

## 8. Next Steps

### *i. Limitations in Experimentation*

1. Number of Testing Images

Although the accuracy of the CNN was evaluated with 3 sets of 1000 images, full testing imagesets often comprise tens of thousands of images, which would have improved the accuracy of testing. Additionally, only a single low contrast imageset was produced. However, if more low contrast imagesets were produced, with varying levels of contrast, it could provide more room for analysis of results.

2. Grayscale

Grayscale images were used in testing due to the fact that they were much simpler to deal with, only having one colour channel. However, most CNNs (and the CNN used in testing) are trained on colour images, each of which have 3 colour channels: red, green, and blue. This means that the filters these CNNs possess have been specifically trained to deal with 3 channel images, and not the single channel that grayscale possesses (Nicholson, 2020). Ideally, the experiment would be redone using colour images to avoid this discrepancy.

3. CNN

The CNN used in this paper was an online CNN API. Although this makes results more easily replicable, it also comes with limitations. First of all, the CNN used was not trained on the same dataset used in this paper's testing. The issue with this lies in the fact that it is possible for the CNN to return a "correct" classification, but for the actual text to be different. For example, an image in the dataset possesses the label "stone", while the CNN returned the label "rock" for that image. The testing program would count this as a wrong label, due to the fact that they are not exactly the same, resulting in a massive drop in accuracy. This issue could be mitigated by training a CNN on the same dataset used for testing (separating images for testing and training), although that would require a great deal more time, computational power, and storage. Using multiple CNNs would also help improve the accuracy of testing results, removing possible outliers, though this too would require more resources.

### *ii. Further Exploration*

Histogram equalization is just one method of contrast enhancement, with its own advantages and shortcomings. Plenty of other contrast enhancement methods exist, some of which try to make up for the deficiencies in HE, like Adaptive Histogram Equalization (AHE), which tries to only operate on smaller segments of the image (Asamoah et al., 2018), or Contrast-Limited Adaptive Histogram Equalization (CLAHE), which both operates on smaller segments of the image, and also limits the amount of enhancement done, to try and keep smaller details visible (Asamoah et al., 2018). To further extend this investigation, it would be interesting to explore how these methods fare against each other in terms of affecting the accuracy of a CNN image recognition AI.

## 9. Bibliography

Asamoah, D., Oppong, E. O., Oppong, S. O., & Dansu, J. (2018, October). Measuring the Performance of Image Contrast Enhancement Technique. https://www.ijcaonline.org/archives/volume181/number22/asamoah-2018-ijca-917899.pdf.

Brownlee, J. (2019, July 5). *A gentle introduction to computer vision*. Machine Learning Mastery. https://machinelearningmastery.com/what-is-computer-vision/.

Capper, P. (2006). *Hawkes Bay NZ* [Photograph]. https://upload.wikimedia.org/wikipedia/commons/0/08/Unequalized_Hawkes_Bay_NZ.jpg

Cburnett. (2008). *8x8 pixel subimage used as an example for JPEG* [Digital Illustration]. https://commons.wikimedia.org/wiki/File:JPEG_example_subimage.svg

Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020, April). *Mathematics for machine learning*. Mathematics for Machine Learning. https://mml-book.com.

Dodge, S., & Karam, L. (2016, April 21). Understanding How Image Quality Affects Deep Neural Networks. https://arxiv.org/pdf/1604.04004.pdf.

Easwaran, S. (2009). *Simplified teaching and understanding of histogram equalization in digital image processing*. Simplified Teaching And Understanding Of Histogram Equalization In Digital Image Processing. https://www.semanticscholar.org/paper/Simplified-Teaching-And-Understanding-Of-Histogram-Easwaran/49b8852b55576732f895ead4e24c715c2f7372f6.

Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (2003). *Intensity histogram*. Image Analysis - Intensity Histogram. https://homepages.inf.ed.ac.uk/rbf/HIPR2/histgram.htm.

IBM. (2020). *Neural Network Diagram* [Digital Illustration]. https://1.cms.s81c.com/sites/default/files/2021-01-06/ICLH_Diagram_Batch_01_03-DeepNeuralNetwork-WHITEBG.png

Jarekt. (2008). *Greyscale frequency histogram of Equalized Hawkes Bay NZ* [Digital Illustration]. https://commons.wikimedia.org/wiki/File:Equalized_Histogram.svg

Jarekt. (2008). *Greyscale frequency histogram of Unequalized Hawkes Bay NZ* [Digital Illustration]. https://commons.wikimedia.org/wiki/File:Unequalized_Histogram.svg

Krukar, M. (2021, August 25). *The role of contrast in ability of human vision*. MkrGeo. http://www.mkrgeo-blog.com/the-role-of-contrast-in-ability-of-human-vision/.

Nicholson, C. (2020). *A beginner's guide to convolutional neural Networks (CNNs)*. A.I. Wiki. https://wiki.pathmind.com/convolutional-network.

NPS. (2008, August). *Understanding Bit Depth*. Conserve O Gram. https://www.nps.gov/museum/publications/conserveogram/22-01.pdf.

Pawar, D. (2018, August 14). *Improving performance of convolutional neural network!* Dipti Pawar. https://medium.com/@dipti.rohan.pawar/improving-performance-of-convolutional-neural-network-2ecfe0207de7.

Peregud, I. (2020, August 19). *The most exciting applications of computer vision across industries*. InData Labs. https://indatalabs.com/blog/applications-computer-vision-across-industries.

Rasulnrasul. (2005). *Plot for illustrating histogram equalization* [Digital Illustration]. https://commons.wikimedia.org/wiki/File:Plot_for_illustrating_histogram_equalization.svg

Rizvi, M. S. Z. (2020, February 18). *CNN image Classification: Image classification USING CNN*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/.

Sanderson, G. (2017). *Neural Networks*. 3Blue1Brown. https://www.3blue1brown.com/topics/neural-networks.

Stewart, M. (2016, February 26). *Simple introduction to convolutional neural networks*. Towards Data Science. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac.

## 10. Appendices

### *i. Appendix A*

A naive implementation of the histogram equalization process.

```
import cv2 as cv
import numpy as np


def histogram_eq(image_name):
    # read the image
    original_img = cv.imread(image_name);

    # convert to grayscale
    img = cv.cvtColor(original_img, cv.COLOR_BGR2GRAY)

    # calculate amount of pixels and channels
    channels = 256
    pixels = img.shape[0] * img.shape[1]

    # generate an array of size 256 to accomodate all 256 possible intensity values (8-bit)
    cumulative_intensity_hist = np.zeros(channels)

    for y in range(0, img.shape[0]): # this for loop runs for every row of pixels in the image
        for x in range(0, img.shape[1]): # this loop runs for every column of pixels in the image

            # image[y][x] returns the intensity value of the pixel at that column and row
            for intensity in range(img[y][x], channels):
                # to generate the cumulative intensity histogram, bins greater than and including the bin of the
                # pixel's intensity will be incremented by one
                cumulative_intensity_hist[intensity] += 1

    # generate an array of size 256 to accomodate all 256 possible intensity values (8-bit) for the lookup table
    lookup_table = np.zeros(channels)

    # calculate the lookup table with the cumulative intensity histogram
    for intensity in range(0, channels):

        # use the formula to calculate the adjusted value
```

```
        adjusted_val = (channels/pixels) * cumulative_intensity_hist[intensity] - 1

        # set the lookup table value to the adjusted value if it is over 0
        if adjusted_val >= 0:
            lookup_table[intensity] = round(adjusted_val)

        # it is 0 by default so no need to set the lookup table value to 0

    # adjust the intensities in the image
    for y in range(0, img.shape[0]): # this for loop runs for every row of pixels in the image
        for x in range(0, img.shape[1]): # this loop runs for every column of pixels in the image
            img.itemset((y, x), lookup_table[img[y][x]])

    cv.imwrite("he_image.jpg", img)
```

### *ii. Appendix B*

The programs used to run the testing in this investigation.

1. A program to convert all images into grayscale

```
import cv2 as cv
import numpy as np
import os

source_dir = 'C:/Users/chane/Downloads/EE images/1000'
output_dir = 'C:/Users/chane/Downloads/EE images/1000_g'
for img in os.listdir(source_dir):
    img_orig = cv.imread(f'{source_dir}/{img}')
    # convert to grayscale
    grayscale = cv.cvtColor(img_orig, cv.COLOR_BGR2GRAY)
    cv.imwrite(f'{output_dir}/{img}', img_g)
```

2. A program to lower the contrast of grayscale images

```
from PIL import Image, ImageEnhance
import os

src = 'C:/Users/chane/Downloads/EE images/1000_g'
destination = 'C:/Users/chane/Downloads/EE images/1000_0.2'

for img in os.listdir(src):
    image = Image.open(f'{src}/{img}')

    enhancer = ImageEnhance.Contrast(image)
```

```
    factor = 0.15
    img_enhanced = enhancer.enhance(factor)
    img_enhanced.save(f'{destination}/{img}')
```

3. A program to histogram equalize images

```
import cv2 as cv
import numpy as np
import os

source_dir = 'C:/Users/chane/OneDrive/Documents/school files/EE\proj/venv'
output_dir = 'C:/Users/chane/OneDrive/Documents/school files/EE\proj/venv'

for img in os.listdir(source_dir):

    image = cv.imread(f'{source_dir}/{img}', 0);
    equ = cv.equalizeHist(image)
    cv.imwrite(f'{output_dir}/{img}', equ)
```

4. The program used to make requests to the API and record them in a JSON. The ‘src’ variable was changed depending on the imageset being tested

```
import cv2 as cv
import numpy as np
import requests
import json
import os

def tag(image_path):
    api_key = 'acc_66f11f4492250e3'
    api_secret = '0a5ee31e0860fc345ffc7f89cfc0093a'
```

```
    response = requests.post(
        'https://api.imagga.com/v2/tags',
        auth=(api_key, api_secret),
        files={'image': open(image_path, 'rb')})
    return response.json()

def setup_json():
    responses = {}
    with open('labels_txt.json') as f:
        labels = json.load(f)
        for key in labels:
            responses[key] = []
    return responses

def format_response(json):
    response = json['result']['tags'][:5]

    format = []
    for item in response:
        keypair = {}
        keypair[item['tag']['en']] = item['confidence']
        format.append(keypair)
    return format

# low contrast dir
src = 'C:/Users/chane/Downloads/EE images/1000_0.2'

# histogram equalized dir
# src = 'C:/Users/chane/Downloads/EE images/1000_he'

# grayscale dir
# src = 'C:/Users/chane/Downloads/EE images/1000_g'

responses = setup_json()

for img in os.listdir(src):
    try:
        result = tag(f'{src}/{img}')
        print(img[:-4])
        responses[img[:-4]] = format_response(result)
        print(responses[img[:-4]])
    except:
        pass

with open('low_contrast.json', 'w') as f:
    f.write(json.dumps(responses, indent=4))
```

5. On occasion, the API would take too long to return values, so the image ID would be left empty.

   To remedy this, a “fix” program was written to re-request the API and fill the empty spaces.

```
import cv2 as cv
import numpy as np
import requests
import json
import os

def format_response(json):
    response = json['result']['tags'][:5]

    format = []
    for item in response:
        keypair = {}
        keypair[item['tag']['en']] = item['confidence']
        format.append(keypair)
    return format

def tag(image_path, api_key, api_secret):
    response = requests.post(
        'https://api.imagga.com/v2/tags',
        auth=(api_key, api_secret),
        files={'image': open(image_path, 'rb')})
    return response.json()

def find_unmatched(labels):
    unmatched = []
    for key in labels:
        if len(labels[key]) == 0:
            unmatched.append(key)
    return unmatched

def fix_unmatched(src, api_key, api_secret, filepath):
    results = ''
    with open(filepath, 'r') as f:
        results = json.load(f)

    unmatched = find_unmatched(results)
    print(unmatched)

    while len(unmatched) != 0:
        for img in unmatched:
            try:
                fp = f'{src}/{img}.jpg'
                response = tag(fp, api_key, api_secret)
                print(img)
                results[img] = format_response(response)
```

```
            print(results[img])
        except:
            pass

    unmatched = find_unmatched(results)

  with open(f'{filepath}_fix.json', 'w') as f:
    f.write(json.dumps(results, indent=4))

# low contrast
src_lc = 'C:/Users/chane/Downloads/EE images/1000_0.2'
api_key_lc = 'acc_66f11f4492250e3'
api_secret_lc = '0a5ee31e0860fc345ffc7f89cfc0093a'
lc_results = 'low_contrast_results_reformat.json'

fix_unmatched(src_lc, api_key_lc, api_secret_lc, lc_results)

# histogram equalized
src_he = 'C:/Users/chane/Downloads/EE images/1000_he'
api_key_he = 'acc_9988fd927555e07'
api_secret_he = 'e1ffc4f522e8646c9831fae85b75ebc7'
he_results = 'he_results.json'

fix_unmatched(src_he, api_key_he, api_secret_he, he_results)

# grayscale
src_g = 'C:/Users/chane/Downloads/EE images/1000_g'
api_key_g = 'acc_4236451543209cd'
api_secret_g = '80ad18ed933d305b109e895d5770b1fd'
g_results = 'grayscale_results.json'

fix_unmatched(src_g, api_key_g, api_secret_g, g_results)
```

6. A program to evaluate the amount of correct labellings each imageset received

```
import json

lc_path = 'lc_results_fix.json'
he_path = 'he_results_fix.json'
g_path = 'grayscale_results_fix.json'
labels_path = 'labels_txt.json'

def load(path):
  with open(path, 'r') as f:
    return json.load(f)
```

```
lc = load(lc_path)
he = load(he_path)
g = load(g_path)
labels = load(labels_path)

results = {}

lc_labels = 0
he_labels = 0
g_labels = 0
num_labels = 0
lc_confidence = 0
he_confidence = 0
g_confidence = 0

for key in labels:
    for label in labels[key]:
        print(label)
        for prediction in lc[key]:
            print(prediction)
            for item in prediction:
                if item.lower() == label.lower():
                    lc_labels += 1
                    lc_confidence += prediction[item]
        for prediction in he[key]:
            print(prediction)
            for item in prediction:
                if item.lower() == label.lower():
                    he_labels += 1
                    he_confidence += prediction[item]
        for prediction in g[key]:
            print(prediction)
            for item in prediction:
                if item.lower() == label.lower():
                    g_labels += 1
                    g_confidence += prediction[item]
        num_labels += 1

results['lc'] = [lc_labels, lc_confidence/float(lc_labels)]
results['he'] = [he_labels, he_confidence/float(he_labels)]
results['g'] = [g_labels, g_confidence/float(g_labels)]
results['num_labels'] = num_labels

with open('final_results.json', 'w') as f:
    f.write(json.dumps(results, indent=4))
```