Abstract

Content Based Image Retrieval, otherwise known as ‘CBIR’ is a method used to extract visually similar images from a large database, based on the features of a given query image. This project explores and presents a modified approach to CBIR by using artificial neural networks, as opposed to classical computer vision techniques. The method presented in this report utilises a Siamese neural network paired with a variation of One Shot Learning, which is trained on pairs of dissimilar images in order to extract image feature vectors. Visual similarity is then deduced by utilising a distance function that compares the distance between the two feature vectors returned by the network and this will therefore provide a measure of similarity based on the image content learned by the network.

1 - Executive Summary

The aim of this report is to present the history and current advancements in the field of Content Based Image Retrieval (CBIR) with the experimental use of Artificial Neural Networks. CBIR Systems are utilised in order to retrieve images based on features extracted by a computer vision algorithm, which can then be compared using a function to deduce image similarity. Modern commercial approaches to CBIR have been increasing in popularity, in 2001 Google launched their image search engine which revolutionised the way we search for images on the internet. CBIR systems allows a disconnect between the end user and the search query, with modern approaches allowing computers to automatically label and identify an image based on its features alone. Further to this, they are also incredibly useful for situation where there is a lack of image metadata, which would semantically describe the image, allowing a system to infer characteristics of the image based merely on its content.

Classical approaches to CBIR relied heavily on the use of embedded geometric and spatial data such as colour, texture, shape and edges, which would be able to describe an image. This data was segregated into two predominant classes, *primitive features* such as those described geometrically and through colour and then *logical features* such as the physical identity of the objects in given regions, both of which had to be separable by an algorithm in order to correctly classify an image.

A more recent approach for extracting features from images requires the use of a convolutional neural network (CNN), as outlined by Koch et al. [23] which is essentially a collection of various layers that all link together in order to extract information from a given input. These can be applied to images in order to extract important features such as colour, texture, shape and edges which are the four main constituents of an image when being processed by a computer. The main downfall of a convolutional neural network is its need for a vast amount of training data for each subset it is required to classify and in many cases, there is a sparse presence of data to begin with, which makes training these types of network rather difficult.

In order to target these issues, many novel approaches have emerged, namely One-Shot-Learning as outlined by Fei-Fei et al. in their report titled ‘One-Shot Learning of Object Categories’ [22]. This report outlines a technique that requires only a small subset of data per class in order to distinguish similarity between both seen and unseen data. The ability to train a convolutional neural network with a much smaller subset of data opens doors to a variety of applications that would usually not be possible, for instance within areas of medicine where there is a lack of data to begin with.

Our network is based on the work of Koch et al. in their paper titled ‘Siamese Neural Networks for One-shot Image Recognition’ [23], whereby we will implement a variation of a standard CNN known as a *Siamese Network*. This architecture utilises two identical networks that take two images as input and process them in order to determine their relative ‘similarity’ by utilising a function that compares the distance between the extracted feature vectors for each image. The results displayed by our network were rather surprising, performing adequately to begin with, but didn’t display a terrific amount of improvement as the amount of time trained increased.

2 - Literature Review

2.1 - Classical Approaches of CBIR

In the past, CBIR techniques required manual feature extraction in order to correctly deduce visual similarity, but due to advancements within machine learning, this can now largely be avoided. Initially, non-metadata based CBIR systems were utilised to aid the sorting and searching of large image databases, without the need for information manually stored in text or annotations, which was likely input by a human. These systems were developed to allow information to be gathered from an image by inspecting pixel content alone, allowing the inference of different features.

2.1.1 - Colour

Many older CBIR techniques utilise colour histograms in order to extract the colour content distribution of an image [1]. The colour content distribution can be analysed between images to denote their colour similarity measure. Many of these techniques are largely similar and act by implementing a variation of basic techniques. A user can retrieve an image by specifying proportions of specific colours within a given threshold, the most basic of these techniques is known as histogram intersection. This technique returns the number of pixels from the input histogram that have corresponding pixels of the same colour in the target image histogram, allowing the user to deduce a colour content similarity measure [2].

A refined method described by Wei-Ying Ma et al. [3] describes the use of a colour codebook to minimise the amount of colours used to represent homogeneous regions/ sections of an image, without detriment to perceptual quality. The approach described in the paper uses the Generalised Lloyd Algorithm in order to quantise the overall colour histogram of a given region. We will not discuss this algorithm in detail here, but rather describe the representation of an image feature through the use of the colour codebook:

(1)

$$\mathbf{f}_c = \left\{ \left(I_j, P_j\right) \mid I_j \in \{1, 2, \ldots, 256\}, 0 \le P_j \le 1 \right.$$
$$\left. \sum_{1 \le j \le N} P_j = 1, \quad \text{and} \quad 1 \le j \le N \right\}$$

Above (1), we describe the colour feature, denoted as $\mathbf{f}_c$, where $\left(I_j, P_j\right)$ denotes the index to the colour notebook and the percentage respectively. In this instance, the colour codebook has a total of 256 colours. This colour feature representation is essentially a compressed version of the full colour histogram for a given region. By computing this metric, we are able to efficiently extract the most prominent colours from a given region, minimising the mean squared error of the original colour content since there is a minute subset of total colours in the codebook in comparison with the total amount of colours in the entire image.

2.1.2 - Shape

Shape data is traditionally very complex to interpret from images, predominantly due to a loss in dimensionality when viewing an image in 2D. The 3D shape data, by definition, is being projected into a 2D plane when represented as an image, causing depth, concavity and other various measures to be semantically challenging to classify via the means of an algorithm.

We can extract shape features in a variety of ways, one example is through the use of regional shape data [4]. Given an image, we can denote the area of a region by analysing the amount of pixels in the region. We can use a simple function that maps pixels within the region as 1 and pixels outside of the region as 0, these are described as $f(x, y) = 1$ and $f(x, y) = 0$ respectively. The area of the region can be then be calculated as:

(2)
$$Area_{region} = \sum_{(x,y)\in R} 1$$

An alternative to using regions is through the use of a centroid distance function (*CDF)* [5]. This is simply a calculation of the distance of the central point of a shape and its respective boundaries.

We can calculate the centroid of a shape in a 2D image as follows:

(3)
$$\text{Centroid} = \begin{Bmatrix} gx = \frac{1}{N}\sum_{i=1}^{N} x_i \\ gy = \frac{1}{N}\sum_{i=1}^{N} y_i \end{Bmatrix}$$

In (3), gx denotes the centre along the x axis of the shape and gy denotes the centre along the y axis - we then simply compute the sum of all N points (x_i, y_i) and then average them.

Finally, we can then calculate the distance to the boundaries from the centroid, where $x(n)$ is the x axis boundary and $y(n)$ is the y axis boundary:

(4)
$$r(n) = [(\mathrm{x}(n) - gx)^2 + (y(n) - gy)^2]^{\frac{1}{2}}$$

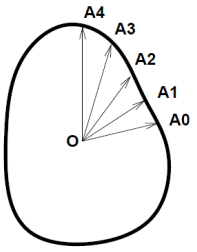


Figure 1 (left) - A visual representation of centroid distances to the boundaries of a shape in 2D [6].

2.1.3 - Edge Detection

Edge detection is one of the most useful forms of feature extraction for CBIR Systems, since it returns incredibly valuable information. Barrow et al. [7] discusses some of these interpretations of edges which can be used to determine scene illumination variation, distinguish between different materials and segment images by separating their foreground and background.

Edge detector algorithms consist of a collection of mathematical approaches to detect pixels within an image that are affected by brightness discontinuities. For instance, a two lines of pixels which sharply change their respective brightness could indicate a difference in depth and thus an edge is present. There are a variety of methods used for edge detection, but they can all be segregated into two main categories: *search-based* and *zero-crossing* [8].

Zero-crossing methods use a second order derivative which is calculated from the image, usually the Laplacian operator, but we will not delve into detail with these methods.

Instead, we can look at an example of search-based techniques. These methods use a measure of edge strength, usually calculated by using a first order derivative, followed by calculating a directional local maxima to classify the direction in which the edge is facing - usually the gradient direction is sufficient.

(5)
$$\nabla F = \left[\frac{\delta F}{\delta x}, \frac{\delta F}{\delta y}\right]$$

Above (5), we see the gradient operator, which is incredibly useful in calculating the direction at which the image function F changes, which is represented as an angle θ, described below (6).

(6)
$$\theta = \tan^{-1}\left[\frac{\delta F}{\delta y} / \frac{\delta F}{\delta x}\right]$$

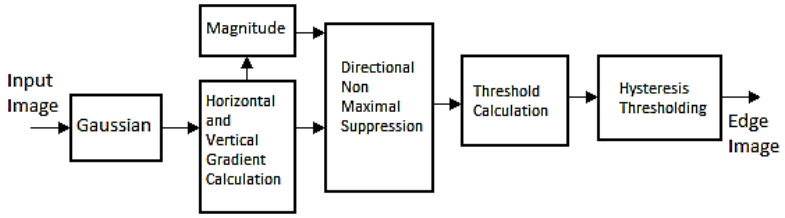
One incredibly useful application of the gradient operator is within the Sobel operator [9], which is a kernel (a small matrix, usually 3x3 or 5x5) that is convolved over the image in order to approximate the horizontal and vertical derivatives.

(7)
$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{F} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{F}$$

The kernels above (7) denote the gradient of the horizontal $\mathbf{G}_x$ and the vertical $\mathbf{G}_y$ by convolving each kernel over the image function $\mathbf{F}$, which in turn produces a map of edges for the image. The Sobel Operator is incredibly simplistic and very easy to implement into a variety of systems, it also gives the orientation of the edges by applying (6) as above [10].
The main issue with the Sobel Operator is the fact that noise causes a detrimental effect on performance, namely, as noise within the image increases, the accuracy of the edges degrades significantly.

A separate approach that targets these issues is the Canny Edge Detection algorithm. This algorithm is part of the aforementioned search-based category of algorithms, but unlike the Sobel Operator, this is a far more complex algorithm. It uses methods such as gaussian blurring to remove noise, non-maximum suppression for removing any regions which do not constitute a real edge and hysteresis thresholding that removes edges below a certain threshold. By thresholding, the signal-to-noise ratio is drastically improved as only strong, well defined edges remain after the computation. The main disadvantage is the fact the computations take far longer and can cause tremendous slowdowns if applied to large image databases [10].

Figure 2 (below) - Canny Edge Detection algorithm pipeline [11].



2.1.4 - Texture Based Approaches

Texture is a marginally different form of feature that can be extracted from an image, many other features rely on geometric data alone, such as edge detection, colour etc. but extracting texture features relies more on spatial characteristics of image regions. There are two main approaches to defining texture within an image, we can either use a *structural* approach or a *statistical* approach [12]. Structurally, we define texture as a set of texels, these are essentially texture pixels which make up a texture map over a region of an image, these texels can be aligned in arrays to form textures.

In fact, we can actually use the previously described edge detection techniques in order to infer some form of texture feature. The feature we are able to extract is known as the busyness of a region of pixels [12].

Suppose we have a region of N pixels, and we apply the aforementioned edge detection techniques and for each pixel p we return two values; the gradient direction, denoted $dir(p)$, and the gradient magnitude, denoted $mag(p)$, both of which can be calculated with (5) and (6). We can then define the *edgeness per unit area* as

(8)

$$F_{edgeness} = \frac{|\{p \,|\, \mathrm{Mag}(p) \geq T\}|}{N}$$

*w*here T is some threshold value.

Further to this, we can also use histograms to represent both the busyness and orientation of a given region and then analyse their respective similarities to each other.
We can describe each normalised histogram as $H_{mag}(R)$ and $H_{dir}(R)$ for the magnitude and direction histograms respectively. From this we can now accurately give a quantitative textural description of a region R as follows:

(9)
$$F_{\text{magdir}} = \Big(H_{\text{mag}}(R), H_{\text{dir}}(R)\Big)$$

In order to deduce whether two regions, or images for that matter, are similar, we must introduce the notion of a *distance metric*. These metrics are incredibly useful in returning the similarity of any given feature vector, allowing them to be generalised not only to texture, but for any pair of vector-valued image features. There are three main distance functions, namely Euclidian, Manhattan or L1 and Cosine similarity [13]. In fact, we can actually represent Euclidian and Manhattan distance as instances of another metric known as the Minkowski distance, defined as follows

(10)
$$d_{Minkowski}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1\ldots n} \Big| x_i - y_i \Big|^p \right)^{\frac{1}{p}}$$

where p is some integer > 0.

Interestingly, when $p = 1$ we have the formula for the Manhattan (L1) distance and when $p = 2$ we have the formula for the Euclidian distance. So, if we let $p = 2$, and let the two input vectors **x** and **y** be two n-valued histograms as mentioned above, we get the L1 distance between the two textural feature vectors and thus can infer how similar the two regions are:

(11)
$$L_1\left(H_1, H_2\right) = \sum_{i=1}^{n} \Big| H_1[i] - H_2[i] \Big|$$

2.2 - CBIR Techniques using Machine Learning

When discussing machine learning methods for CBIR, it's important to note that the key to any successful CBIR system is a precise and accurate feature extraction algorithm. Many of the underlying techniques used within machine learning implementations are synonymous to traditional CBIR methods, involving data pre-processing, feature extraction and evaluation. Yet instead of devising an algorithm to extract image features ourselves, we train a neural network to infer image properties and learn feature repres-

entations by providing samples of training and test data. Our implementation relies heavily on the use of Convolutional Neural Networks, which are explained below.

2.2.1 - Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialised type of network structure that are used to learn on data with a grid-like topology [14]. For this reason, images are incredibly well suited for this type of network. CNNs function by replacing standard matrix multiplication techniques with another linear operation, known as convolution. They are incredibly efficient with matrix transformation based tasks, making them brilliant for image data. Kernels are convolved within the hidden layers of the neural network, which can be used as edge detectors or blur filters, similar to those mentioned in previous sections [14]. CNNs can also utilise the back-propagation algorithm [16] in order to increase the network performance as it trains. Put simply, it steps back through the network and adjusts each individual weight proportionally to the amount of error it contributes to the network itself, over time, this tunes the network and will begin to produce better outputs.

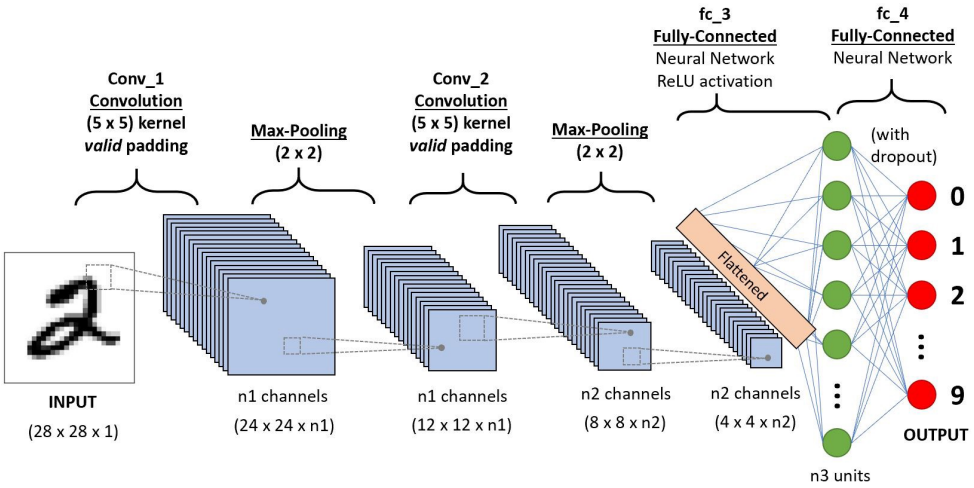
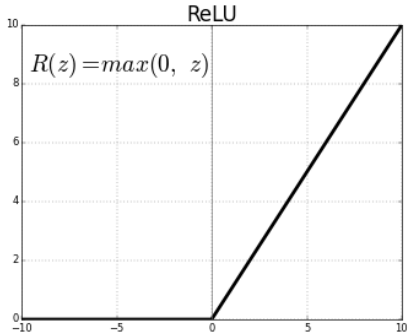


Figure 3 (right) - an example of standard architecture used in a deep convolutional neural network [15].

As you can see in Figure 3, an image is taken as input to the network, the image is then convolved using a 5x5 kernel and then passed to a *pooling* layer, this essentially reduces the spatial size of the representation of the data in order to reduce computation within the network. More convolutions / max-pooling layers can be added or removed dependent on the architecture you choose to adapt. We can see each layer description is represented as a triple ($pixels_x$, $pixels_y$, $operations_n$), for example the *Conv_1* layer is a 5x5 kernel that convolves over the image forming a 24x24 output and this is repeated $n1$ times within this layer.



After each layer has finished its computation, it's passed through some function that transforms the summed weighted input from the layer into the activation of the next layer. There are a variety of activation functions [17], including Sigmoid, Hyperbolic Tangent and ReLU; we will discuss ReLU further as it's very commonly used within CNNs:

Figure 4 (above) - The ReLU activation function [18].

Here, the ReLU function takes in the input z and transforms it to 0 if it is < 0 or 1 if it's $\geq$ 0. This leads to sparse activation of neurons, since it's removing any values that are below 0, resulting in a 'lighter' network.

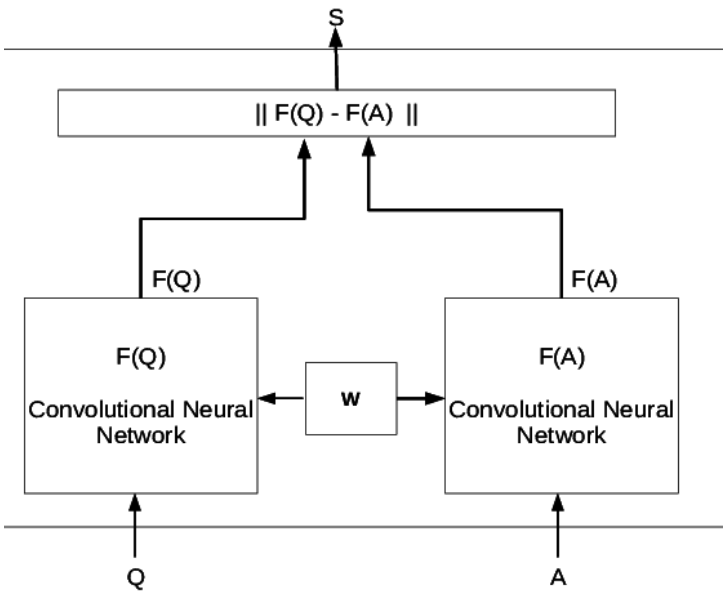
However, in order to effectively train traditional deep CNNs, we usually require a lot of labelled data, and sometimes this is not attainable, for instance in the medical field with new disease etc. so this can be challenging, however, there are approaches such as one-shot-learning that I will go on to discuss later, in order to tackle this issue.

2.2.2 - Siamese Neural Networks

In 1994, the first implementation of a Siamese Neural Network was proposed by Jane Bromley et al. at AT&T [19]. The network was devised as a means of verifying handwritten signatures and used a pair of identical networks, hence the name 'siamese network'. This was revolutionary because the proposed model would extract a feature vector from a pair of arbitrary signatures and then deduce their respective similarities - in this case, the cosine distance metric is utilised. In order to verify a signature, an extracted feature vector is compared with a preexisting database of signatures and will be able to display all other similar signatures. If the input signature is within a certain similarity threshold value, it is deemed as valid, else it is an invalid signature.

Siamese networks, even nowadays are rather sparse and are not widely used, but their popularity is slowly gaining traction. The network pair used at AT&T uses a time-delay network, which is largely outdated and instead, the common implementation nowadays is to utilise a pair of CNNs in order to commence training, as described in SigNet [20].

Figure 5 (below) - An example of t
he revised Siamese Network architecture [21].



As you can see, the each network shares weights and parameter updates throughout the training of the network. The two identical CNNs are joined and their feature vectors $F(Q)$ and $F(A)$ are input into a similarity metric, in this instance the Euclidian Distance is calculated as the similarity measure.

It must be noted that Siamese Networks do not directly classify images, instead they just tell us how similar two images are. This is fundamentally important for a CBIR system, as it allows us to view all images that are visually similar from the database, by learning and comparing the image features automatically through the use of CNNs.

2.2.3 - Feature Extraction in Siamese Nets with One Shot Learning

In the absence of vast amounts of training data, traditional CNNs would struggle to learn different representations of classes and perform well at prediction of images. However, we can use a method known as 'One Shot Learning' in order to train our network using only one image to train our network, with remarkable performance. This method was introduced by Fei-Fei et al. [22], whereby it was hypothesised that; 'once a few categories have been learned the hard way, some information may be abstracted from that process to make learning further categories more efficient.'

The predominant theory surrounding this paper relies on a Bayesian approach to learning, and in the paper they discuss the notion of determining whether a certain item is present within an image or not. We can compare the probability of only background clutter being present vs. Instances where the item is present. To do this we require a model of the aforementioned item that is gathered from a set of training images.

We can now begin to formulate the problem as follows:

Let I be the query image, which may contain an example of the foreground category, denoted by O_{fg}. The alternative is that it contains background clutter / noise belonging to a generic background category denoted by O_{bg}. I_t is the set of training images that we have used as the foreground category. Now, the decision of whether this query image I has the foreground object or not can be written in the following way using a bayesian framework [22]:

(12)

$$R = \frac{p\left(O_{fg} \mid I, I_t\right)}{p\left(O_{bg} \mid I, I_t\right)} = \frac{p\left(I \mid I_t, O_{fg}\right)p\left(O_{fg}\right)}{p\left(I \mid I_t, O_{bg}\right)p\left(O_{bg}\right)}$$

Let R denote the ratio of the class posteriors, if $R >$ T (some threshold value) we assume that input image an instance of the item. This expression is a decision about whether I contains the image in the foreground, with an absence of background clutter.

The beauty of this approach is that we need only 1 image denoting the item we are searching for in order to train, leading to an incredibly valuable approach to implement into a pre-existing siamese network.

3 - Methodology

CBIR systems have a plethora of uses, predominantly within database searching on image websites or hospitals for example. For our CBIR implementation, we will be using a technique described by Koch et al. [23] which utilises a Siamese Neural Network with a One Shot Learning based approach. Our network will be implemented in Python using the Tensorflow and Keras Libraries.

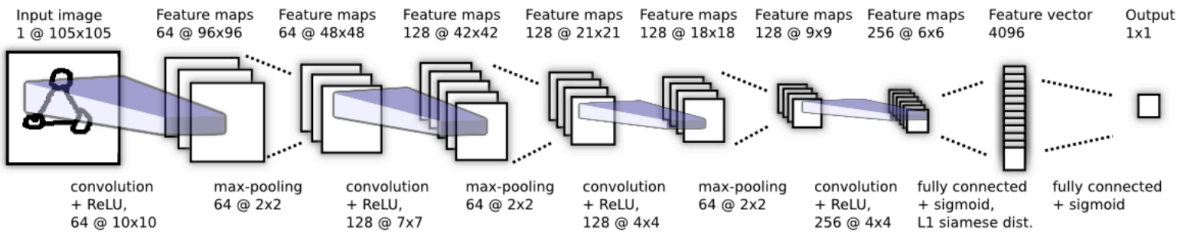
3.1 *- Assumptions*

The motivation for the implementation is as follows: we will be implementing an abstracted version of the architecture, using grayscale images of a variety of items. We can assume for this reason that a use case could include a doctor / medical researcher with a database of images of a variety of different scans of patients, with a variety of illnesses and diseases. The doctor could then use this system to return visually similar scans, x-rays for example, from patients who have similar presentations, in order to aid their diagnosis by following up and viewing patient history etc. This data will likely be sparse, as medical data is rather difficult to attain for research and thus the motivation for using a One Shot Learning based approach. This will provide a level of abstraction for the network to allow a simplistic implementation for testing and we will assume the training images have no associated semantic data, aside from the category of images they reside within.

3.2 *- Siamese Neural Network Implementation*

Since Siamese Neural Networks are uncommon, there is a sparsity of papers. As mentioned above, we will be basing our implementation on that of Koch et al. [23] as it provides a robust and well known approach. A model of one branch of the siamese network is shown in Figure 6 (below).

Figure 6 (below) - An example of the network architecture [23].



This network will receive (x_1, x_2) as input, where x_1 and x_2 are two images, and will return a two feature vectors describing each image. From these two outputs, we can begin to infer if the images are in fact the same by utilising a distance function. The function noted in the paper is the Manhattan Distance (L1 Distance) and so we will be implementing this into our network. Once this metric has been calculated, it is passed to a

sigmoid activation function which will constrain the value of the L1 distance to be between 0 and 1 and this will denote the relative similarity of the image pair (x_1, x_2).

More formally, the network structure can be defined as mentioned in [23]:

In terms of network tuning, we used the ADAM optimiser [24] with a learning rate of 0.00006 and binary cross-entropy as our loss function due to the binary nature of our problem i.e. a similarity value will be returned between 0 and 1. Also, the fact it makes the network sparser will resulting in a much more time-efficient loss convergence rate.

Since we have a Siamese network, we take two inputs, one to denote each image and these are fed into the input layer of each network, which has a relative resolution of (105 × 105) pixels. For all of the following convolutional layers, we ensure they all utilise the ReLU activation function and a small L2 kernel regulariser value, this is around 0.0002. Directly following the input layer, there is the first convolution layer of size 64 (10 × 10), followed by a 64 (2 × 2) max pooling layer to reduce the size of the output and then another convolutional layer of size 128 (7 × 7), followed by another identical 64 (2 × 2) max pooling layer. Next, a 128 (4 × 4) convolution layer is followed by a 64 (2 × 2) max pooling and then the final convolution layer is added, with size 256 (4 × 4). We then flatten the output of this layer which is then followed by a dense layer of size 4096 with another sigmoid activation function for the weights. Finally, there is the distance function that is utilised by the network, in order to implement this using Keras, we must use a Lambda layer [25] which allows us to calculate the Manhattan (L1) distance of the feature vectors. Finally, the output of this is combined into a dense layer of size 1 using a sigmoid activation function, which will then finally return the similarity value for the two input images.

3.2.1 - K-Way One Shot Learning Algorithm Implementation

For our network, we are assuming that there are a distinct lack of labels for the data and that for each category, there is not likely to be an abundance of training images. For this reason, we have opted to use the One Shot Learning approach, as discussed in **2.2.3**.

However, upon further research, there may be a better implementation [26]; because of this, we will propose the following amendments to the current One Shot Learning approach:

1. We will likely be using ‘Few-Shot Learning’ (K-Way One Shot Learning) as we may have more than one item of data per category, but this is not guaranteed. Though we may have more data, it is not a requirement for the network, so long as we have at least one piece. This is essentially identical to the One Shot approach, but with potential for more data per class.
2. We can utilise the modified approach in order to test and validate our network in realtime, as it trains on our inputs.

So, the modified approach is formulated as follows:

Let us assume we have 16 categories of data and we wish to test how our network is performing as it trains, we can use 16-Way One Shot learning for example. Essentially what this means is we are testing image pairs in order to deduce their similarity and comparing them against their actual similarity measure. In the case of 16-Way One Shot learning, we compare our query image to 16 other images, each of a different class aside from one, returning a set of 16 different similarity scores for the query image. Since all 16 images are different, aside from one which is identical, we know that one of the images should hypothetically have the highest similarity measure out of all of the images. If this is not the case, then the model has predicted incorrectly and this can be fed back to the network for future iterations. It is also important to mention that for K categories of data, we do not have to restrict ourselves to use K-way one shot learning, we can use essentially any value of K, so long as it lies between 2 and the amount of categories, in this case = K. It is trivial that using a smaller value for K will result in higher chances of scoring correctly, but this will also not be as accurate when compared to using a larger value of K, so we must choose a compromise between performance and computation time.

3.2.2 - Hyper Parameter Selection

Our justification for hyper parameter selection relies solely on the research conducted by Koch et al. [23] for this specific network. Since we are using a deep convolutional neural network, training it on the CPU alone will cause a major detriment performance and time taken to train the network. Because of this, we must use a GPU, yet cannot reap full benefit without sufficient GPU optimisation knowledge within Keras. Because of this and due to the time-constraints paired with this project and a lack of, we will be using the hyper parameter values outlined in the aforementioned paper:

1. We randomly initialised the weights and biases of our network according to the following constraints outlined by Koch et al:

Assume a normal distribution for both weights and biases.

(13)

$$w_{layer} \sim N\left(0.0, 1e^{-2}\right)$$
$$b_{layer} \sim N\left(0.5, 1e^{-2}\right)$$

2. Our learning rate was set to 0.00006 and we utilised the ADAM optimiser for this, as mentioned in **3.2**.

3.3 - Data Pre-Processing

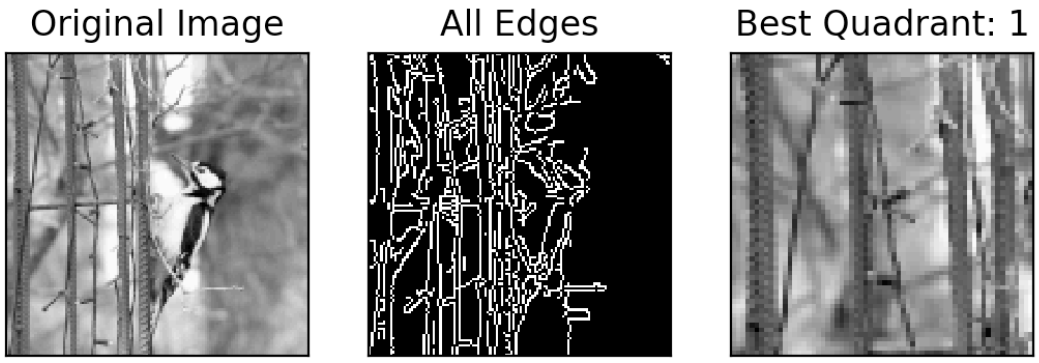
When looking at the network architecture, we can see that the input layer requires an image of dimensions (105 × 105) pixels. Most image databases do not contain images of this size and so we must pre-process this data in order for it to fit the requirement of our outlined network. Because of this, we must also ensure that our images are not less than the required (105 × 105) pixels as this would require upscaling the image and producing varied results due to the introduction of image artefacts.

So, for all images that require resizing in order to function correctly, we hypothesised an approach to crop the images based on regions of interest using the Canny Edge Detector mentioned in **2.1.3**:

1. If not already, convert the desired image to grayscale.
2. Perform edge detection using the Canny approach, returning an array of all points that are assumed to be an edge.
3. Divide the image into four quadrants, denote these $Q_1, Q_2 \ldots$ etc.
4. For each image quadrant, count the amount of pixels present.
5. Select the quadrant with the most points. This is our region of interest (ROI).
6. Crop the image outward from the ROI if necessary (i.e. region is smaller than the required (105 × 105) pixels.)

Unfortunately, after some testing and experimentation, this generated unreliable results. In some images with a distinct foreground and background separation, this performed adequately, generating some useable results, but images with a large amount of variation essentially ended up with regions that are unusable.

Figure 7 (below) - An example of hypothesised cropping algorithm.



Instead, we decided to use a trivial approach of downscaling each image to the required (105 × 105) pixels by using a centre crop. Most of the items in our dataset were centred and this provided the simplest approach for grayscale images; this also alleviated the risk of losing data by simply downsampling the resolution and further compressing the image by reducing resolution.

Aside from the cropping algorithm, any colour images were converted to grayscale using the Python CV2 library and each class of data was reshaped to contain a uniform distribution of images by randomly selecting a subset of images from each class.

3.4 - Image Similarity for Retrieval

In order for the network to actually function as an image retrieval system, we must be able to extract image predictions from the network. We are able to use Keras' inbuilt $model.predict()$ function in order to return the probabilities of the matching images.

We simply pair our query image with other random images within our dataset as follows

(14) $$[(query, image_1)\ (query, image_2)\ (query, image_3)\ (query, image_4)]$$

where $image_i$ is the i^{th} image in the dataset, for some user-defined value i.

This will generate a list of probabilities of the similarity of our query image with respect to the dataset image by using the L1 distance metric described in **3.2**, i.e. for 4 comparison images:

(15) $$\begin{bmatrix} S_1 & S_2 & S_3 & S_4 \end{bmatrix}$$

4 - Tests, Analysis and Evaluation

4.1 - Network Performance Measure

In order to accurately quantify the performance of our network as it trains, we will be using K-One Shot Learning in order to test our network. Images are decided in advance if they are similar within the dataset, because of this we use the category of image as the benchmark of similarity. We are able to denote the performance by specifying the amount of 'guesses' the network gets correct out of a number of one-shot verification tasks.

The performance measure is formulated as follows

(16)
$$\text{performance}\ (\%) = \left(\frac{p_{correct}}{k_{total}} \right) \times 100$$

where $p_{correct}$ is the amount of correct guesses of similar images with respect to the query image and k_{total} is the total number of guesses by the network.

4.2 - Image Datasets

For both training and validation we decided to use two datasets of varying size. The first dataset that was utilised is the Omniglot dataset [27], the reasoning for this is that it is a great benchmark for the network for a few reasons:

1. The dataset was purpose built for one-shot learning tasks, with a limited number of examples per category, making it very useful to evaluate the performance of our network.
2. The dataset was used in the paper by Koch et al. [23] but with far more images appended to it through the use of affine distortion of each character. We thought it would be interesting to test the accuracy of the network on this dataset without the added extra data, to test its performance on a sparser version of the dataset.
3. There are is a fixed number of character examples per category, which makes the data slightly uniformly balanced as compared to the other dataset we are using, this will be a good test for the overall network similarity accuracy.
4. The images are all grayscale and already fit the (105 x 105) pixel input requirement.

The second dataset used was the Stanford Dogs dataset [28] which is a subset of the massive ImageNet database [29] consisting of millions of images. This was manually divided into an 80:20 train:validation split, with random categories assigned to each.

Figure 8 (below) - Full dataset descriptions.

Dataset	Total Categories	Total Images
Omniglot [45]	1,623	32,460
Modified Stanford Dogs [46]	120	20,580

4.3 - Network Testing

As mentioned before, in order to effectively train our network without exceptionally long wait times we must utilise a powerful GPU in order to streamline this process. There are a variety of different online machine learning cloud providers that offer GPU support including Google Cloud, Amazon Web Services, Microsoft Azure and Vast.ai. However, Google now offers a free service with a GPU known as 'Google Colaboratory', it gives us limited, but suitable access to an Nvidia Tesla K80 GPU which is deemed to be around 5-10x faster than a CPU alone [30].

We tested the network with a variety of different parameters, all of which are discussed beneath each test result. With regards to hyper parameter variation, we will change that batch size, K-value for One Shot Learning, the number of One Shot tasks to evaluate and the number of iterations for the network.

Each time the network is run, it is evaluated after a specified number of iterations using K-One Shot Learning on a separate test set, the best outcome of this is then noted and is the performance benchmark for that specific network configuration.

4.3.1 - Omniglot Dataset Benchmark

Since this is the dataset used within the paper by Koch et al. it was deemed the most appropriate to be a benchmark of our network. A few tests were run with different parameters to see how these affected performance. The dataset has a total of 50 Alphabets containing a total of 1623 Characters, 30 alphabets were used for training and the remaining 20 alphabets were used for validation. In total there were around 33,000 images used between the train and validation sets.

Omniglot - Test 1

We first conducted a small test run of our network, to check it's functionality on a small number of iterations. In this case, we used a total of 1000 iterations with a batch size of 32. Along with this, the network was evaluated every 200 iterations and we used K-One Shot Learning with a value of $K = 4$ along with a value of $K_{total} = 10$.

The network performed very well with a performance measure of 100% on our test set. In total, it took 3.1 minutes to train and converged to a loss value of 2.16.

This loss value could have been optimised further, but the network did not run for long enough to converge to a smaller value.

Omniglot - Test 2

For the next test of the network, we wanted to check it's performance on a larger number of iterations and now increase the K_{total} value for a more accurate representation of performance. In this case, we used a total of 20,000 iterations with a batch size of 32. Along with this, the network was evaluated every 200 iterations and we used K-One Shot Learning with a value of $K = 20$ along with a value of $K_{total} = 20$.

The network yet again performed very well with a performance measure of 83.2% on our test set. In total, it took 76.8 minutes to train and converged to a loss value of 0.215.

Omniglot - Test 3

The third and final test of the network was a somewhat in-between configuration, to see how well the network would perform with half of the number of iteration as before and see how this affected its performance. In this case, we used a total of 10,000 iterations with a batch size of 32. Along with this, the network was evaluated every 200 iterations and we used K-One Shot Learning with a value of $K = 20$ along with a value of $K_{total} = 20$.

The network yet again performed worse than in our second test, but still surprisingly well, with a performance measure of 78.0% on our test set. In total, it took 62.8 minutes to train and converged to a loss value of 0.244.

By halving the number of iterations, we can see there is a 6.6% performance decrease, but the network converged to a similar loss value around 22.3% faster than before. So we can evidently see there is a balance that needs to be decided based on network training speed and the performance of the network.

In comparison to the findings of Koch et al. [23] our network performed slightly worse than their network on around 30,000 images to train. Our network performance was 83.2% after 20,000 iterations, compared to 90.61% in the paper. This could have been attributed to a few factors, but the main being the fact the network could have been run for more iterations in an attempt to increase performance.

4.3.2 - Modified Stanford Dogs Dataset

With this dataset, a couple of tests were performed, yet again with different parameters to see how these affected performance. The dataset has a total of 20,580 images, split 80:20 between training and validation sets.

Modified Stanford - Test 1

For the first test, we used a total of 10,000 iterations with a batch size of 32. Along with this, the network was evaluated every 200 iterations and we used K-One Shot Learning with a value of $K = 4$ along with a value of $K_{total} = 5$.

The network performed rather poorly when compared to the Omniglot dataset with a performance measure of a mere 40% on our test set. In total, it took 76.8 minutes to train and converged to a loss value of 0.711.

Modified Stanford - Test 2

For the second test, we now wanted to see if increasing the total number of iterations would result in a better performance score. Alongside this, we wanted to see if we could further reduce our training loss value and so we modified a few other parameters too. We used a total of 20,000 iterations with a batch size of 32. We also decided to change the interval of evaluation to 500 iterations, and we used K-One Shot Learning with a value of $K = 10$ along with a value of $K_{total} = 5$.

The network performed somewhat better when compared to the first test, resulting in a performance of 62.1% on our test set. Surprisingly, the time taken to train the network on this configuration with more iterations was in fact reduced by 32.3 minutes for a total of 44.5 minutes of training and converged to a loss value of 0.708.

Surprisingly, this variant of the parameters lead to a similar loss value in less time, but yielded higher performance overall. This is likely due to the ratio of the rate of evaluation and total iterations. In this instance, we evaluated every 500 iterations of the total 20,000 iterations, this gives a ratio of 1:40, whereas in the first test, we evaluated every 200 iterations of the total of 10,000 iterations which gave a ratio of 1:50. So, in essence we evaluated the network less times over the total number of iterations in test 2 which would have likely positively impacted the training
time.

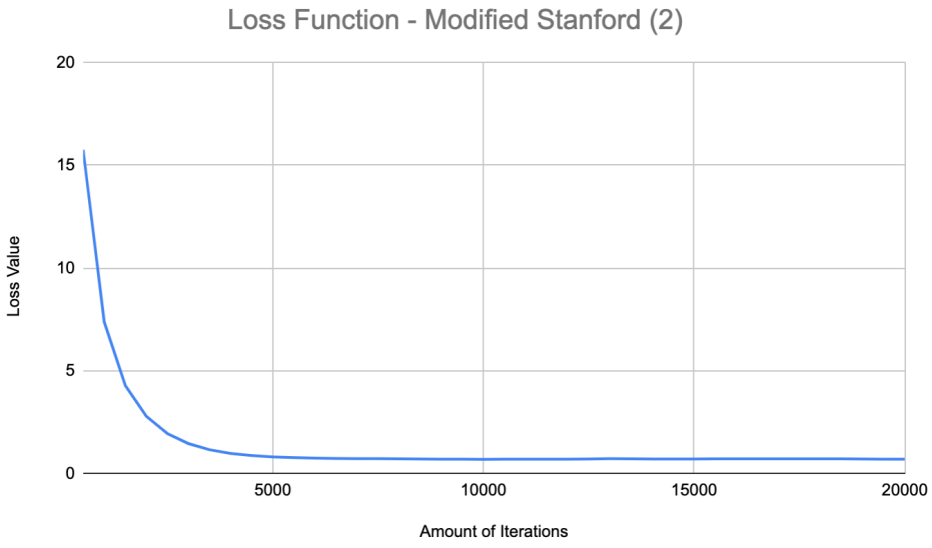


Figure 9 (right) - Final loss function over 20,000 iterations for Modified Stanford Test 2.

We can clearly see here in *figure 9* (above) that after around 3000 iterations, the network loss does not seem to improve. This could be due to a number of reasons but could be predominantly due to the chosen dataset and our cropping algorithm described in **3.3** which may have had an impact on the network's ability to minimise its total loss farther than ≈ 0.7. Our hypothesis for increasing the performance of our network was correct in predicting that the number of iterations increased will bring positive impact to the overall performance of the network, but what was rather unexpected was the change in total train time when reducing the amount of evaluations done by the network, even with double the amount of iterations present. We can therefore deduce that as a proof of concept, the use of K-One Shot Learning and Siamese networks does yield some promising results. Further tweaking and optimisation of the network would be required but it performs adequately at $\approx 62\%$ success rate when compared to random guessing which would have yielded 25% and 10% performance for each of the Modified Stanford tests respectively.

5 - Conclusion and Future Work

In this report, we have presented a modern and cutting-edge approach to the CBIR problem, using Siamese Neural Networks and a modified one-shot learning approach. The results attained show an $\approx$ 62% success rate for multiple classes which has been shown to be far superior to randomised guessing. These results however do not match that of Koch et al. [23], but this may be due to differences in datasets, pre-processing and network optimisation. In comparison, this approach provides a modern method of image retrieval without the need for semantic image labels or metadata, which provides a solution toward many industries with a lack of sufficient data (i.e. medical research.)

While this implementation provides a somewhat decent method of retrieval, there are associated social, ethical and legal issues that have to be addressed. A major example of an ethical and legal issue lies within the field of medicine, if this type of system were to be used in order to aid the diagnosis of a patient, it's current success rate alone would not be a feasible method in order to treat these patients. A recent article discussed the factors associated with computer aided diagnosis of breast lesions [31] and concluded that specific visual presentations of these specific types of lesions are more likely to produce false results: 'Larger benign lesions, the presence of lesion calcifications, and high degrees of vascularity are likely to show false-positive results. Smaller malignant lesions and the absence of calcifications are likely to show false-negative results.'

Given more time, it would have been beneficial to create a generalised version of this network to function with a variety of different images. Our network was hindered by the fact only grayscale images were being used, by implementing the use of colour and other features present in many of the classical CBIR techniques, we may have yielded better performance and potentially built a network that generalises features better.

6 - Bibliography

[1] - J. Eakins and M. Graham, “Content-based image retrieval,” Content-based image retrieval by Eakins, John, Graham, Margaret, 1999.

[2] - I. W. J.. Aloimonos, R. Bajcsy, D. H. Ballard, C. M. B. D.H.. Ballard, I. Biederman, B. A. W. D.H.. Brainard, D. Chapman, J. A. Feldman, Y. Y. J.A.. Feldman, D. A. Forsyth, S. A. S. G.J.. Klinker, A. J. D. J.J.. Koenderink, M. D. Z. P.. Lennie, P. P. J.. Malik, B. W. L.T.. Maloney, W. T. N. J.H.R.. Maunsell, R. C. Nelson, K. P. R.. Ohlander, K. S. J.. Rubner, A. Treisman, and A. L. Yarbus, “Color indexing,” International Journal of Computer Vision, 01-Jan-1988.

[3] - W.-Y. Ma and B. S. Manjunath, “NeTra: A toolbox for navigating large image databases,” 01-May-1999.

[4] - J. Liu and Y. Shi, “Image Feature Extraction Method Based on Shape Characteristics and Its Application in Medical Image Analysis,” SpringerLink, 20-Aug-2011.

[5] - R. C. Gonzalez, R. E. Woods, and S. L. Eddins, Digital image processing using MATLAB, Vol.2: Gatesmark Publishing Knoxville, 2009.

[6] - P. Arjun et al., “Compact centroid distance shape descriptor based on object area normalization - IEEE Conference Publication,” 08-May-2014.

[7] - H. G. Barrow and J. M. Tenenbaum, “Interpreting Line Drawings as Three-Dimensional Surfaces,” 1981.

[8] - H. S. Bhadauria, A. Singh, and A. Kumar, “Comparison between Various Edge Detection Methods on Satellite Image,” CORE, 2013. [Online].

[9] - R. F. et al., “Sobel Edge Detector,” Feature Detectors - Sobel Edge Detector, 2003.

[10] - D. Kim, “Sobel Operator and Canny Edge Detector ECE 480,” 2013. [Online].

[11] - B. Crnokic et al., “Comparision of Edge Detection Methods for Obstacles ...,” 2016. [Online].

[12] - L. G. Shapiro and G. C. Stockman, “Section 7 - Texture,” in Computer vision, Upper Saddle River, NJ: Prentice Hall, 2001.

[13] - S. Ontañón, “An Overview of Distance and Similarity Functions for Structured Data,” NASA/ADS, 2020. [Online].

[14] - K. P. Murphy, M. Kejriwal, C. Knoblock, and P. Szekely, “Deep Learning,” The MIT Press, 2016. [Online].

[15] - S. Saha, “A Comprehensive Guide to Convolutional Neural Networks ,” Medium, 17-Dec-2018. [Online].

[16] - “Backpropagation,” Backpropagation - ML Glossary documentation, 2017. [Online].

[17] - A. F. M. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” 07-Feb-2019. [Online].

[18] - K. Sarkar, “ReLU : Not a Differentiable Function: Why used in Gradient Based Optimization?,” Medium, 31-May-2018. [Online].

[19] - J. Bromley et al., “Signature Verification using a 'Siamese' Time Delay Neural ...,” 1994. [Online].

[20] - S. Dey, A. Dutta, J. I. Toledo, S. K. Ghosh, J. Llados, and U. Pal, “SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification,” arXiv.org, 30-Sep-2017. [Online].

[21] - A. Das et al., “Together we stand: Siamese Networks for Similar Question ...,” Jan-2016. [Online].

[22] - L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” NYU Scholars, 20-Aug-2015. [Online].

[23] - G. Koch et al., “Siamese Neural Networks for One-shot Image Recognition,” 2015. [Online].

[24] - D. P. Kingma and J. Ba, “ADAM: A Method for Stochastic Optimization,” arXiv.org, 30-Jan-2017. [Online].

[25] - K. Team, “Keras documentation: Lambda layer,” Keras. [Online].

[26] - H. Sankesara, “N-Shot Learning: Learning More with Less Data,” FloydHub Blog, 16-Aug-2019. [Online].

[27] - B. Lake, “brendenlake/omniglot,” GitHub, 13-Feb-2019. [Online].

[28] - Aditya Khosla et al. “Stanford Dogs dataset for Fine-Grained Visual Categorization.” , Stanford University [Online].

[29] - ImageNet. [Online].

[30] - “Deep Learning Benchmarks of NVIDIA Tesla P100 PCIe, Tesla K80, and Tesla M40 GPUs,” Microway, 30-Jan-2017. [Online].

[31] - J.-Y. W. et al., “Computer-Aided Diagnosis of Solid Breast Lesions With Ultrasound: Factors Associated With False-negative and False-positive Results,” Journal of ultrasound in medicine : official journal of the American Institute of Ultrasound in Medicine, Dec-2019. [Online].