A Comparison of Haralick, LBP and CNN Features for Content Based Image Retrieval

Dawit Tirore

March 2019

Contents

1 Introduction

Medical images are an important diagnostic tool for physicians and the technology for taking these images keep improving. Common types of medical image types include magnetic resonance imaging (MRI), computed tomography (CT), photon emission tomography (PET) and many more. Interpreting these images however requires domain expertise and it's labor-intensive. Because of this, Computer Aided Diagnosis (CAD) is sometimes used to assist the physician in making a diagnosis. One type of CAD is Content-Based-Image retrieval (CBIR) which based on a query image, tries to retrieve images that are relevant to this from a large collection of images. However, currently most hospitals use a system for organizing medical images that's called Picture Archiving and Communication System (PACS).

PACS uses Text-Based-Image-Retrival (TBIR) which means that images are indexed by metadata describing the image such as age, patient ID, date and a manually annotated description of the image [1]. There are some disadvantages of using this technique. One is that text can't easily capture visual information about the picture, such as color, texture and shape. Another is that manually annotating these images is labour-intensive. A third is that the difference of perception for physicians in describing the image can lead to inaccuracy when retrieving images.
Because of these reasons there is a high demand for CBIR techinques as it doesn't have any of the above mentioned problems. CBIR does however comes with its' own set of challenges, which we'll return to later in this introduction.

By using CBIR in CAD the workload for doctors can be decreased by providing more consistent and reliable analysis of medical images. Given an image database with diagnosis information, CBIR based CAD aims to retrieve images that are most similar to a query image. Imagine a scenario where an undiagnosed patient get's an MR scan and the CBIR queried on that MRI returns 9 images that are diagnosed as strokes. The doctor could then act fast in determining if the patient was indeed having a stroke.

1.1 Challenges of CBIR

CBIR is based on two challenges. Feature extraction and feature indexing. The first deals with how we efficiently and accurately represent images. The second deals with how we efficiently search large databases for similar images based on these representations. The usual setup for a CBIR system has the following structure. First the CBIR is queried with an image. This image is then reduced to one or more vectors describing high-level features of the image through the feature extraction phase. The CBIR system has access to an image database where feature representations of these images already is computed. The second phase searches in this database, returning the k most relevant images, based on some distance measure between the feature vector of the query image and the relevant images. Often, these image databases are enormous, so smart search techniques have to be employed. This is the problem of feature indexing. In this section we'll describe some of the methods that are used in feature extraction and feature indexing

1.1.1 Feature Extraction

Feature extraction is the problem of how we represent an image as a set of features. Usually this set of features is denoted as a feature vector (or several feature vectors) and we want this feature vector to have a low dimensionality to save space and computation-cost. We also want our feature vector representation that yields a smaller distance between representations of similar images. The representation must also yield a larger distance between dissimilar images. Determining features that fulfill these criteria is especially difficult in the medical image domain because of the problem described as "large intra-class variation and small inter-class variation" [2]. What this means is that medical images can vary by a lot in spite of lying in the same class. It also means that medical images that don't lie in the same class can look very similar.
Feature extraction methods can be split into two categories. Hand-crafted features and learned features. We'll look into both.

Hand-crafted features are usually modeled by experts, where each feature models some specific information about the image, that is relevant to the expert. This method dominated the field before deep learning became widespread in this area. There are some disadvantages of using hand-crafted features. The method requires expert knowledge and even still, expert knowledge usually doesn't work when the database is very large because there can be many outliers not following the standardised rules. Hand-crafted features can also be very time-consuming to design and slow to execute. Finally many hand-crafted methods only specialize in specific image data [2].

One example of a very successful method that uses hand-crafted features, is SIFT. SIFT stands for Scale-Invariant-Feature-Transform. The name also highlight an important aspect of feature representations, namely *invariance*. SIFT is a scale- and rotation-invariant feature extraction method. This means that feature representations of an image, will be similar even as it's rotated or changed in scale (increased or decreased in size). We will later see how another method, LBP, achieves rotation-invariance. SIFT first detects keypoints, which are points of interest, and then computes features based on descriptors of these keypoints. They are found using two steps. First, the image is convolved with a gaussian kernel, the difference between this image and the original is then computed. The second step detects which areas of the image that have the largest difference between the original and the blurred image. SIFT then uses a method called Histogram of Oriented Gradients (HOG) to create a descriptor for the keypoint and these descriptors are then used to create the features. We skip lightly over the technical details of SIFT as this isn't our focus area. The main takeaway from how SIFT computes features is that it computes *localised* features. The features are based on keypoints, rather than the entire image [16].

An alternative method to localized features, which also uses hand-crafted features, is using Haralick texture features on an entire image. Haralick texture features are 13 features proposed by Haralicks et al in [3] that are computed on a grey-level-coocurrence-matrix (GLCM). In the theory section we'll give a more detailed account of what a GLCM is, but for now it suffices to know that it's a matrix that encodes the relationship between pixel values in a grey-level image. Given a grey-level image, after its' GLCM is computed, [3] proposed 13 operations to perform on the GLCM, which all try to capture a high level feature of the image. These features include Energy, Entropy, Correlation and more.

Learned features can be implemented with Deep Learning. It's performance is depended on training data and therefore requires large sizes of it to achieve a high accuracy. A clear advantage of using learned features is that expert knowledge isn't needed, and the vast size of image data available can be used to train good models.
Deep learning based learned features can be split in two categories. Features based on supervised learning or features based on unsupervised learning. Supervised learning means that you train your neural network using training data that is labeled. In the case of brain MRIs this could be MRIs that are labelded with a diagnosis, age or other property relevant to the given task. Unsupervised learning is training, where you don't have access to labels.
One type of Deep Learning method that is commonly used (and most often used as a supervised learning method), is Convolutional Neural Networks (CNNs). CNNs work by first having a convolution layer, where a number of kernels are convolved on the original image, producing a number of new images. This is usually followed by a pooling layer that downsamples these new images. Usually this is followed by some fully connected layers that's finally passed to the output layer where the classification happens. The details of this will be explained in the theory section.

Supervised deep learning methods have shown high accuracy when used but their drawback is that they need labeled image data. By the nature of medical image data, it can only be annotated by physicans and is therefore expensive and in many cases labels just aren't available on medical images. To avoid these limitations, unsupervised deep learning methods have been suggested. Examples of these are deep stacked auto-encoders (SAEs) and Boltzmann machines, both of which we won't discuss in this paper.

1.1.2 Feature indexing

Once feature extraction is done, each image is represented by a feature vector. Given a query image, feature indexing can now be seen as a nearest neighbour problem by computing distances between query image and database images and retrieving the images with smallest distance [2] . When dealing with large scale databases, a linear search through the entire collection, isn't feasible and therefore other techniques must be used. Some of these are vocabulary trees and hasing methods. We will now look into hashing methods.
Hashing methods work by reducing the feature vectors to short binary codes based on some hashing function. The nearest neighbours are then found by computing distance in binary Hamming space instead of the feature space [2]. Given an image I, with a feature vector R of dimensionality n, its' binary code is computed by a set of hash functions $h_1, h_2, ..., h_k$ with some size k. Each hashing function encodes R into one bit. Concatenating their results yields the binary code for the image.
The main challenge when using this method is how to get good hashing functions that respect the original similarities and differences. Broadly speaking there are two types of hashing, data-independent and data-dependent. Data-independent hashing functions are generalised to work on any kind of image. Their drawback is caused by this generalisation. The binary codes are longer and therefore also require more hashing functions than data-dependent hashing. Data-dependent hashing that learns the hashing functions from a training set, have shown to provide equal or better accuracy in their retrieval with shorter codes compared to data-independent hashing [2].

1.2 Scope

In this project we will focus on the feature extraction phase. We will use Haralick texture features, Local Binary Patterns and Convolutional neural networks, to create feature representations MRIs of knees. This will be tested both in 2d and 3d. As a motivating example, we will also test their performance on a dataset of triangles, square and circles to illustrate the setup. Besides the geometric shapes there are 2 datasets of MRI scans of knees that the methods are tested on. All methods have parameters that are tuned during training and the details of this will be described in the method section.

1.3 Evaluation

When the CBIR system is built we want a concrete way to evaluate it's performance and there are some protocols in place that helps with this [2].
Precision, is defined as the fraction of retrieved images that are relevant to the query image.
It can be written as

$$\text{Precision} = \frac{\text{Relevant images} \cap \text{Retrievede images}}{\text{Retrieved images}} \tag{1}$$

Recall is defined as the fraction of relevant retrieved images from all the relevant images in the database.
It can be written as

$$\text{Recall} = \frac{\text{Relevant images} \cap \text{Retrievede images}}{\text{All relevant images}} \tag{2}$$

Recall measures how good the CBIR is at finding all relevant images. A CBIR can have a high recall and low accuracy by retrieving almost all the relevant images plus a bunch of unrelated images. An example of when a high recall would be important is in case an image database has to be reorganized and it depends on the CBIR's ability to find all the relevant images.

To measure effeciency, the CBIR's average runtime is a common measure.
It can be written as

$$\text{AvgTime} = \frac{1}{M} \sum_{m=1}^{M} t_{m,K} \tag{3}$$

Here $t_{m,K}$ expresses the time cost of querying the CBIR with the mth image, retrieving K relevant results [2].

AUC measures how well a classifier can distinguish between classes, when you vary the threshold for classificiation. AUC stand for Area Under the Curve, and is the are aunder the ROC curve. The ROC curve is the curve you get when you plot Recall (also called TPR) on the y-axis and $\frac{FP}{TN+FP}$ (called FPR) on the x-axis, as you vary a threshold. If we have a classifier that given 5 images returns the array [0.2,0.3,0.6,0.7,0.8], then setting a threshold of 0.5 groups the first two into one class and the rest into another. Varying this threshold changes TPR and FPR and the AUC gives a measure for how good the classifier is across all thresholds. An AUC of 1 is perfect but an AUC 0.5 is. As an example see This curve shows that this is a good classifier as it with a low error rate can classify most of its' samples correctly.

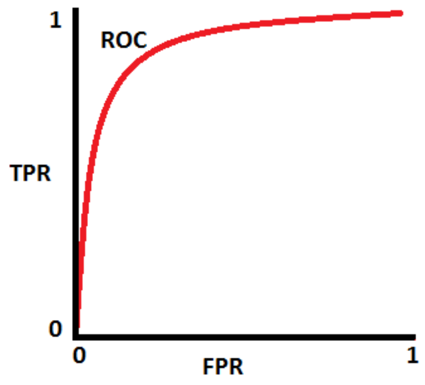
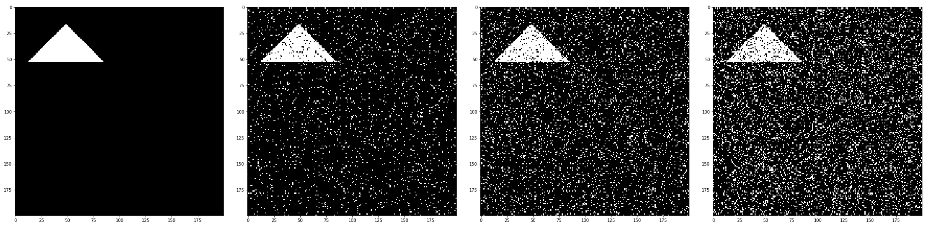


Figure 1: Two circular neighborhoods https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

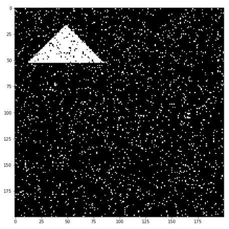
2 CBIR for triangles, squares and circles

As a motivating example, let's remove ourselves from the domain of medical images and take a look at some simple geometrical shapes.

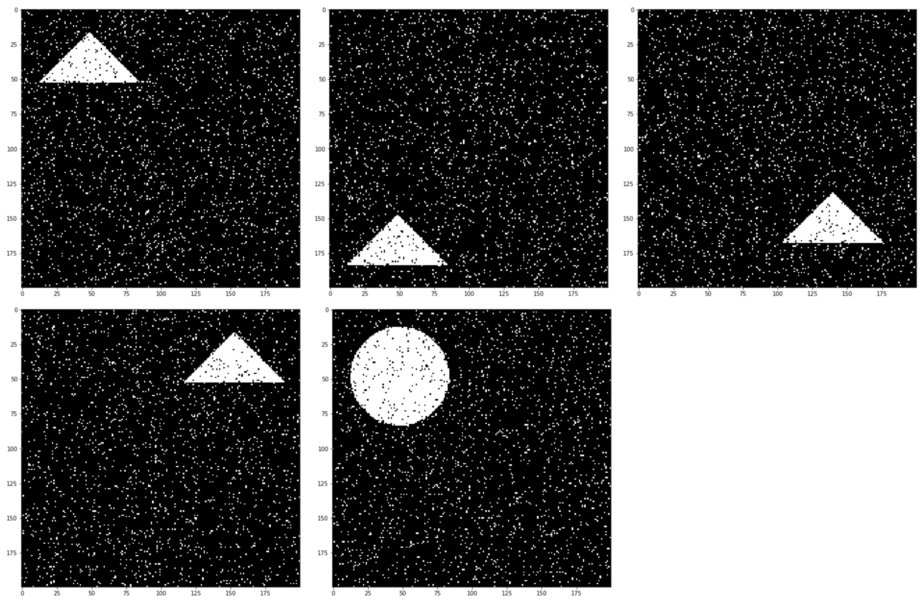
We'll use a dataset consisting in equal amounts of circles, triangles and squares. These shapes vary in both added noise and their placement. There are 4 noise levels, and they are shown below with increasing levels from left to right:



The levels going from left to right are 0%, 10%, 20% and 30%. These percentages indicate how many of the pixels that are assigned a value of 255 or 0 (with equal probability), with the rest unchanged. Furthermore, there are 4 placements of the shapes, one for each corner. This amounts to $4 \cdot 4 \cdot 3 = 48$ distinct images.
We'll now test a naive feature extraction method, the average of the image. As a distance measure we'll use the euclidean distance.
We will use this image as our query image:



This triangle image has a noise level of 10%.
For efficiency we'll begin with computing the feature representation of all images in our collection and save these representations for later use. Let's call the collection of these saved features our "feature database". Concretely it's an array of floating points, where the i'th index holds the image average of the i'th image in our original image collection. We can now query our database with a query image. This process consists of first calculating the feature representation of the query image, ie. it's average, which is 19.794. After this, we iterate through our feature database, calculating the euclidean distance between the average of the query image, and each entry in the feature database. These distances are tupled along with the index of the feature in the feature database, used to calculate the distance. Sorting this list in ascending order with respect to the distance returns our method's choice of most similar image in decreasing order. We'll return the first 5, which yields (left-most are most relevant):



The returned images have the following properties:

Shape	Order	noise	placement	image average	distance to query imgage
Triangle	1st	10%	top-left	20.523	0
Triangle	2nd	10%	bottom-left	20.445	0.083
Triangle	3rd	10%	bottom-right	20.681	0.153
Triangle	4th	10%	top-right	20.075	0.453
Circle	5th	10%	top-left	35.069	14.541

To evaluate the results, we determine relevant results to be be retrieved images of the same shape as the shape present in the query image.
We see that the first returned image is exact same image as the query image. This makes sense as the query image is part of the collection and its' distance to its' own mean is 0. In fact the first four returned images are triangles which suggests that a simple a feature as the image average, might be a decent choice for this kind of problem. The placement of the returned images is also varied, indicating that the method isn't sensitive to position. This is good as image relevance isn't determined by position and it's therefore desirable that the method is invariant to position. This also makes sense as the image average isn't affected by the placement of the shape.
The reason we chose to retrieve 5 images is that there in the dataset for any specific shape and noise level only exists 4 images (the four placements). It is therefore interesting to see what is retrieved as the fifth image.
The fifth image is a circle which might seem counter intuitive as there are still 12 triangles in the dataset that weren't retrieved. The distance between the 5th result and the query image is approximately 30 times larger than the difference

between the query image and the 4th result. Since the CBIR choose the circle as the 5th, the 12 other triangles must have an even greater distance. If we take a look at the noise column in the table, we see all retrieved images have a noise level of 10%. These results suggest that this method is more sensitive to the noise level than the shape. The results also suggest that the method only is sensitive to the shape so long that the noise level between the query image and sample image is the same.

3 Theory

In this section, the theory behind the feature extraction methods used in this project will be described.

3.1 Haralick Texture Features

We briefly touched on Haralick Texture Features in the introduction. These are statistical second order features, describing a texture, where a texture in our case is a grey-level image. The features are second-order features because they are computed from the grey level co-ocurrence matrix (GLCM) (which we'll see in a moment), which expresses the relationship between two pixel values. Had the features been computed directly from a texture, it would have been first-order features .
As said the GLCM expresses the relationship between pixel values. More formally, the GLCM is a matrix M in which the entry at M[i,j] denotes the number of occurrences where pixel value i have a neighbour of pixel value j. The direction the GLCM is calculated with respect to, defines what a neighbour is. Let's take a simple example. Here we have a 4×4 grey-level image:

$$\begin{bmatrix} 1 & 2 & 2 & 3 \\ 1 & 1 & 2 & 2 \\ 3 & 1 & 3 & 2 \\ 0 & 2 & 3 & 0 \end{bmatrix} \tag{4}$$

The GLCM is calculated with respect to the direction (0,1) (one horizontal right-shift). This results in a GLCM that is an $N \times N$ matrix where N is the number of possible grey-level values. From the image above we get the following GLCM:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{5}$$

If we now look at the entry M[2,3] we see that it's 2, and this expresses that pixel values of 2 have occurred with a right neighbour of pixel value 3, 2 times. Often this matrix is also normalised such that each entry is divided by the sum of all entries. The normalised GLCM of (5) can be seen here:

$$\begin{bmatrix} 0 & 0 & \frac{1}{12} & 0 \\ 0 & \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ 0 & 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{12} & \frac{1}{12} & 0 \end{bmatrix} \tag{6}$$

We can see that the entries sum to one, and we know that each entry describes the occurrences of a pixel intensity relationship divided by all occurrences. The entries can therefore been seen as probabilities of pixel intensity relationships. This is the process of computing the GLCM of a 2d grey-level image but Haralick Texture Features can also be computed from a 3d grey-level image and the process is very similar. In 3d you define the relative location of the neighbour pixel in 3 dimensions. If we define the direction to be (0,1,0) the resulting GLCM will count pixel value relationships between all pixels p in the 3d image and the pixel directly to the "right" of p.

Usually the GLCM won't be computed both in direction (1,0) and (-1,0) as they compare the same relationship. In 2d there are thus 4 directions the GLCM can be computed with respect to, and in 3d there's 13. It's general practice to calculate the GLCM in all directions (4 for 2d and 13 for 3d) and take the average of these to ensure some rotation invarians.

Haralick texture features compute statistical measures on the GLCM. There are 13 operations that can be performed on the GLCM and each yield a feature. [4] has suggested a grouping of the features into 3 distinct categories, namely contrast, orderliness, and descriptive statistics of the GLCM. In total there are 13 features, so we'll choose one from each category and explain them in detail. In the formulas below, N denotes the length of a $N \times N$ GLCM and i and j denote row indexing and column indexing respectively. P_{ij} denotes an entry in the GLCM at row i and column j, and is called P because we view it as a probability.

3.1.1 Energy

This feature is also called Angular Second Moment and belongs to the orderliness category. The formula is given by:

$$\sum_{i,j=0}^{N-1} {P_{ij}}^2 \tag{7}$$

High values of Energy means that the texture is very orderly. This is the opposite of another feature, Entropy, which measures how random or "chaotic" a texture is. Intuitively the larger the entries in the GLCM, the larger the value of ${P_{ij}}^2$

would be in those entries. Therefore few large entries yields a higher value than many small entries. An entry denotes how many repetitions there are of a specific pattern and therefore textures with a high repetition of the same patterns yields a higher orderliness.

3.1.2 Contrast

Not surprisingly, this feature is part of the contrast category. It's formula is given by:

$$\sum_{i,j=0}^{N-1} P_{ij}|i-j| \tag{8}$$

This formula sums the absolute difference between all combinations of pixel values weighted by the combinations probability. Intuitively for $P_{ij}|i-j|$ to be large, both the difference between i and j, as well as their probability must be large. In other words the image has high contrast when there are many occurrences of pixels with a large difference between them.

3.1.3 GLCM Mean

This feature is part of the descriptive statistics category. It's formula is given by:

$$\mu_i = \sum_{i,j=0}^{N-1} i \cdot P_{ij} \tag{9}$$

Note that μ has i as a subscript. If the GLCM isn't symmetric, μ_i and μ_j won't perform the same operation. μ with j as subscript is given by:

$$\mu_j = \sum_{i,j=0}^{N-1} j \cdot P_{ij} \tag{10}$$

If we as an example assume the GLCM is symmetric, μ_i effectively calculates the mean pixel value of the original image. This can be seen when we define the probability of pixel value i as:

$$P_i = \sum_{j=0}^{N-1} P_{i,j} \tag{11}$$

We can now rewrite the GLCM Mean as:

$$\mu_i = \sum_{i=0}^{N-1} i \cdot P_i \tag{12}$$

It can be seen that μ_i calculates the sum of pixel values weighted by their probability. This is the same as weighing the pixel value by their occurrence and dividing with the total number of occurrences.

A final note on these features is that features within the same group are highly correlated with each other. One example which was previously mentioned was the inverse relationship between Energy and Entropy [4]. This is useful to take into account when deciding which features to use for a given task.

3.2 Local Binary Patterns (LBP)

Local Binary Patterns is another method that can be used for feature extraction. The method was introduced by Ojala et al. [5] and since then, extensions have been proposed to the method. On a high level, LBP describes each pixel p in an image by comparing the grey-level of p with it's surrounding pixels. This comparison results in a compact descriptor for each pixel, in the form of an integer. A histogram of the occurrences of these integers is then created.

More formally, given a position $\boldsymbol{\xi_0}$ of an image *f*, the LBP operator $G_{y,r}\{f\}(\boldsymbol{\xi_0})$ describes the arrangement of the surrounding pixels of $\boldsymbol{\xi_0}$ in the circular neighbourhood $\Upsilon(\gamma, r, \boldsymbol{\xi_0})$ where γ denotes the amount of points in the neighbourhood and r denotes the distance between $\boldsymbol{\xi_0}$ and each point in the circular neighbourhood. If a point doesn't lie exactly inside a pixel centre, its' value will be interpolated. The binary value of $\mathbf{p} \in \Upsilon(\gamma, r, \boldsymbol{\xi_0})$ is 1 if $f(\boldsymbol{\xi_1}) > f(\boldsymbol{\xi_0})$, where $f(\boldsymbol{\xi_1})$ is the pixel value of p, otherwise it's 0.

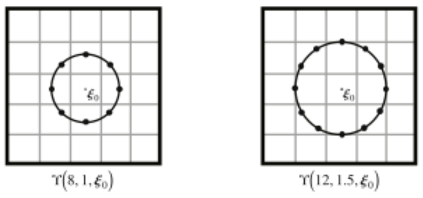


Figure 2: Two circular neighborhoods

In other words, the pixel value at $\boldsymbol{\xi_1}$ is compared with all points in the circular neighbourhood, each comparison yielding a 1 if the neighbour pixel value is greater than the pixel value of $\boldsymbol{\xi_0}$. This results in a sequence of binary values that is evaluated as an integer. Using the LBP operator at each pixel of image *f*, the distribution is represented as a histogram. This histogram is then the descriptor of the image [6].

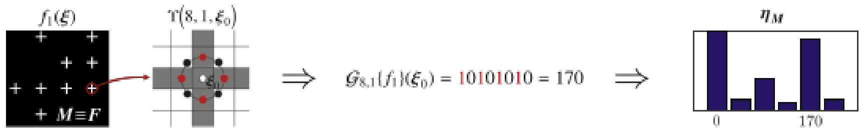


Figure 3: Computing the histogram

Figure 1, 2 and 3, made by [6] illustrates the process of how LBP is computed. Figure 1 shows two different neighbourhoods. The first has a size of 8, and a radius 1. The second has a size of 12 and a radius 1.5. The left side of figure 2 shows how the LBP operator is used at some position $\boldsymbol{\xi}$ of image f_1. The red points and black points highlight higher and lower values respectively, in relation to $f_1(\boldsymbol{\xi})$. The LBP operator's bit sequence and decimal value is also shown. On the right side of the second row, a histogram is shown that describes the distribution of the LBP operator values for the entire image.

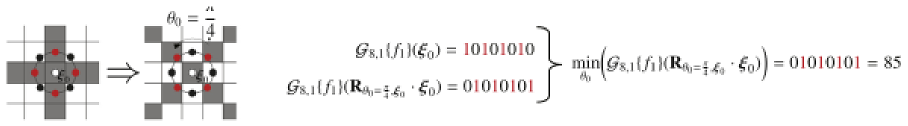

$\theta_0 = \frac{\pi}{4}$
ξ_0
$\mathcal{G}_{8,1}\{f_1\}(\xi_0) = 10101010$
$\mathcal{G}_{8,1}\{f_1\}(\mathbf{R}_{\theta_0=\frac{\pi}{4}}\xi_0 \cdot \xi_0) = 01010101$
$\min_{\theta_0}\left(\mathcal{G}_{8,1}\{f_1\}(\mathbf{R}_{\theta_0=\frac{\pi}{4}}\xi_0 \cdot \xi_0)\right) = 01010101 = 85$

Figure 4: An operation to achieve rotational invarians

The third row shows how rotational invarians can be achieved. The example rotates the region by 45 degrees anti-clockwise and shows the value of the original LBP operator and the rotated LBP operator. These can be mapped to the same binary sequence by bit-wise doing the circular shift in the sequence that results in the smallest integer value.

Uniform LBP is a variant of the LBP, which makes a distinction between uniform patterns and non-uniform patterns. Uniform patterns are defined as having at most two transitions from 0 to 1 or 1 to 0, when traversed circularly. As an example, 0101 is not a uniform pattern as there are 3 transitions but 0100 is uniform. LBP operators with a uniform pattern are each evaulated to the integer value of the bit sequence. LBP operators with a non-uniform pattern are however all mapped to the same value (distinct from any of the uniform patterns).
The resulting histogram can be seen as an n-dimensional feature vector V, where n is the number of bins and V_i is the occurences in the i'th bin. All non-uniform patterns are put in the same bin and this can greatly reduce the length of this vector. Using a neighborhood size of 8, initially the length is $2^8 = 256$ but with uniform patterns it will be 59. (ref: https://en.wikipedia.org/wiki/Local_binary_patterns). One might wonder whether information is lost in the process, but according to Ojala et al, they noticed that non-uniform patterns were rare in their texture images. Using a neighborhood size of 8 and radius 1, uniform patterns accounted for at slightly less than 90 percent of the cases.

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is one branch of Artificial Neural Networks that uses many layers. The CNN is made up of a input layer, an output layer and usually several hidden layers. The input layer is fed some data, which in our case is an image. Some operations are performed on the image, in the hidden layers and finally the data is passed to the output layer where a classification usually happens. In this section we will describe some of the operations that are used in CNNs.
CNNs are inspired from how neurons in the brain function. A neuron can fire

or it can be still, depending on the stimuli provided to it by other neurons it's connected to. In the same way, a layer is comprised by an array of neurons and if we pass a raw image to the input layer, each neuron will contain a pixel from the image. A fully connected layer, will connect each of it's neurons to all neurons in the subsequent layer. Let's assume that the input layer takes a grey-level image of size 10×10 and is followed by a fully connected layer of the same size. Each neuron in the input layer would be connected to 100 neurons, meaning 10,000 connections in total. By, connection I mean a weight and bias. The neuron has some scalar value, and multiplying it with a weight and adding a bias, results in the value that will appear in the neuron it's connected to. This is however a large amount of weights and biases and CNNs try to reduce the complexity of the data. Instead of a fully connected layer, small areas of fx 3×3 of the input layer are connected to a single neuron in the next layer. This reduces the weights and biases and conceptually it is the same as moving a filter across an image, which creates a new down sampled image. I refer to figure 5. As with all the weights and biases connecting neurons in an Artificial Neural Network, their values are initialised to reasonable random values and fitted during training through backpropagtation. We will go into this in a moment.

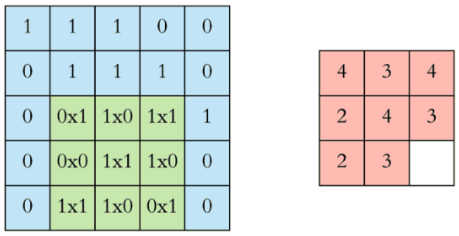


Figure 5: Two circular neighborhoods
source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

As said, moving a filter across an image, creates a new smaller image. This process can be repeated on the smaller image. With each layer having its' own filter that can extract different features of their input. Conceptually earlier convolution layers capture simple patterns like edges. Later convolution layers, capture more complex patterns by building on the defections of the earlier layers.

Beside convolution layers, pooling layers are also used. They reduce their input data by aggregating an area of the input into a single neuron. As an example, max-pooling partitions the image into $n \times n$ regions, extracting the maximal value in each region. Often a region size of 2×2, is chosen. See figure 5 for an illustration that uses 2×2.

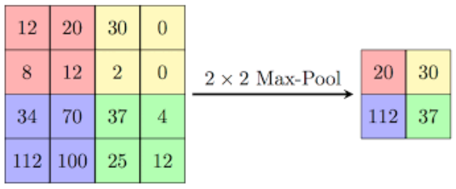


Figure 6: Two circular neighborhoods
source: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Often the output of a neuron is wrapped inside an activation function, like RELU. RELU stands for rectified linear unit and applies the following operation to each neuron

$$RELU(x) = max(0, x) \tag{13}$$

Before RELU became popular, other functions such as sigmoid were used heavily. Activation functions are necesary because it lets the CNN learn nonlinear relationships between its' input and output. This allows the CNN to model more complex relationships than what can be achieved with a linear function [12]. One reason that RELU is often used now is that it doesn't have the problem of "vanishing-gradient". Activation functions like sigmoid or tanh have a gradients that's very close to zero everywhere except the center, which is a problem because the gradients are used to improve the weights and biases during training [13].
Another activation function is SoftMax. Given a vector it converts the values to probabilities so each value is between 0 and 1 and the entries sum to 1. This is commonly used as a final step in an CNN to return valid probabilities for classification. [13]
As said, the weights and biases are initialised to reasonable random values. A CNN can be seen as a function whose parameters is this large set of weights and biases. As an example, let the input be an image and the output a classification. The goal of this CNN is to learning a mapping between an input space and an output space. Let's assume our CNN is trained to do a binary classification task of identifyinf pictures of dogs. The output space is a 2d vector and the output [1,0] denotes a dog, where [0,1] denotes that there isn't a dog in the image. Before training the CNN might output confusing output like [0.7,0.7]. Our goal is to teach the CNN that images of dogs should have an output closer to [1,0] and vice versa.
To do this, you define a cost function that gives a measure of how "wrong" the model is in its' classifications. We can define a simple cost function.

$$E_{total} = \sum \frac{1}{2}(target - output)^2 \tag{14}$$

A low cost function means the model has learned the relationship between input and output well. E_{total} is a function of weights and biases and the gradient of this function expresses the direction of steepest ascent. The negative gradient expresses the direction of the fastest descent. For each of the parameters of E_{total} we can calculate the partial derivative of E_{total} with respect to that parameter. This expresses the rate of change of E_{total} as a function of that parameter. Subtracting the parameter with the rate of change willbring it closer to a local minimum. See figure.

I will now give an example of how weights are computed in a simple neural network, heavily inspired by an explanation given by [14]

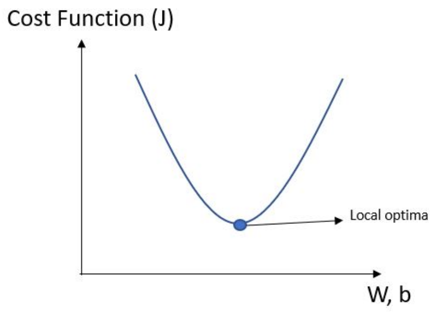


Figure 7: Two circular neighborhoods *source:* https://engmrk.com/error-and-cost-function-for-nn/

Looking at figure x, the red values, are the weights. We distinguish between the net input of a neuron, and the output of a neuron. Net input is the sum of weighted incoming neuron values. The ouput of a neuron is the net input squashed by an activation function. Looking at $h1$ we denote $net_{h_1} = 0.15 \cdot i1 + 0.25 \cdot i2$ and $out_{h_1} = sigmoid(net_{h_1})$. Sigmoid squashes its' input into the range [0,1] [1].

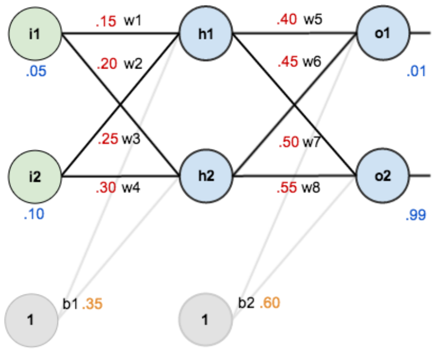


Figure 8: Two circular neighborhoods *source:* https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Training is based on a forward pass and a backwards pass. In the forward pass we pass input to $i1$ and $i2$ and compute the error of the output of out_{o_1} and out_{o_2} by comparing the to the target labels. This is done by the cost function and to decrease the cost function, let us look to w_5 connecting $h1$ and $o1$. We want to know how changes to w_5 affects the cost. What we are interested in is $\frac{\partial E_{total}}{\partial w_5}$. This can be computed with the chain rule [2]. E_{total} depends on out_{o_1} which depends on net_{o_1} which finally depends on w_5. We can therefore write

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} \cdot \frac{\partial out_{o_1}}{\partial net_{o_1}} \cdot \frac{\partial net_{o_1}}{\partial w_5} \tag{15}$$

Decreasing w_5 by $\frac{\partial E_{total}}{\partial w_5}$, moves it closer to a local minimum. Often the rate

[1] $sigmoid(x) = \frac{1}{1+e^{-x}}$

[2] if a variable z depends on y, which again depends on x,then $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$

of change is multiplied with a learning rate less than one, to control the size of the jump when descending.

3.4 Distance measures

In this project we will be using two distance measures. For Haralick Texture Features and Convolutional Neural Network features we will use the euclidean distance. For Local Binary Pattern features we will use J-divergence.

The euclidean distance between two points **p** and **q** who both are included in the coordinate space R^d, is the length of the line conneting them $|\boldsymbol{pq}|$. As both points are part of R^d, we have that $\boldsymbol{p} = (p_1, p_2, ..., p_n)$ and $\boldsymbol{q} = (q_1, q_2, ..., q_n)$. The distance, d, is given by the pythagorean formula:

$$d(\boldsymbol{q}, \boldsymbol{p}) = d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{16}$$

As shown in [7].

J-divergence is based on Kullback-Leibler Divergence (KL-divergence). KL-divergence originates from probabiltity theory and information theory. It is a measure for the difference between two probability distributions $p(x)$ and $q(x)$. This can be used for histograms, as they in their normalised form can be viewed as discrete probability distributions. To explain how KL-divergence is calculated, we must explain the concepts, entropy and cross-entropy.
Entropy is defined as:

$$Entropy = -\sum_{i=1}^{N} P(i) log_2 P(i) \tag{17}$$

The concept originated in the domain of information theory where Claude Shannon was interested in how information could be sent effeciently without losing any information. He defined entropy as the smallest possible average size of lossless encoding of the messages sent from the source to the destination. This can be seen by (14) if we view i as a unique message we want to communicate to someone. In total there are N unique messages. Let's assume we have a history of how often each unique message was sent, meaning we have a probability distribution of the messages. In this example, entropy is the lower limit for the average size of a messages in bits using our probability distribution.
If we rewrite (14) as

$$Entropy = \sum_{i=1}^{N} P(i) \cdot (-log_2 P(i)) \tag{18}$$

We see the sum is comprised of the probability of message i being sent, $P(i)$, multiplied by the negative logarithm of $P(i)$. The negative logarithm of $P(i)$ calculates the minimum size of bits needed to send message i. An example case that shows this is:

$$log_2 N = -log_2(1/N) = -log_2 P \tag{19}$$

If we have N distinct messages, we need $log_2 N$ bits to be able to distinguish them. This can be rewritten as $-log_2(1/N)$. We assume each message is sent with equal probability $1/N$. We can therefore substitute it with P (the probability of the message) so we get the negative logarithm of $P(i)$[3].

Instead of writing (14), entropy can be rewritten as an expectation, with H meaning entropy:

$$H(P) = \mathbb{E}_{x \sim P}[-log(P(x)] \tag{20}$$

Meaning that we both use the probability distribution P to determine frequency of the different messages, and we use it to compute the minimum size of each message x.

Similarly to entropy, there's another measure, cross-entropy:

$$H(P, Q) = \mathbb{E}_{x \sim P}[-log(Q(x)] \tag{21}$$

The only difference is that the probability distribution Q is used to compute the minimum size of each message x, while still using P to determine the frequencies of the messages.
As entropy gives a lower bound for the size of the average message, $H(P, Q)$ can never be lower than $H(P)$.
One example of using cross-entropy is using it as a cost function in training a classifier. An ideal probability distribution for an image of a dog, would have 100% probability of classifying the image as a dog. If we train a model to output a probability distribution when it sees the same image, it might be far from the ideal and this results in higher cross-entropy. The trained model would be "perfect" if the cross-entropy could be minimised until it was equal to the entropy.

KL-divergence is the difference between entropy and cross-entropy:

$$D_{KL}(P||Q) = H(P, Q) - H(P) \tag{22}$$

Which can be written as

$$D_{KL}(p(x)||(q(x)) = \sum p(x) ln \frac{p(x)}{q(x)} \tag{23}$$

[3]This explanation isn't a proof but a way to give some intuition for the formula. Therefore we won't describe the case where the messages have different probabilities

Elegantly said in [8], "The KL divergence measures the expected number of extra bits required to code samples from $p(x)$ when using a code based on $q(x)$, rather than using a code baed on $p(x)$.
If Q = P, D_{KL} is 0, and if Q is very different from P, D_{KL} will be large. KL-divergence is not a distance measure, as it isn't symmetric. $D_{KL}(P||Q)$ isn't necessarily the same as $D_{KL}(Q||P)$. As cross-entropy never is lower than entropy, it's a non-negative measure.

J-divergence is defined as:

$$J(P||Q) = H(P||Q) + H(Q||P) \tag{24}$$

The nice property of J-divergence is that it's symmetric and therefore can be used as a distance measure.

4 Datasets

Three datasets have been used in this project.

The first dataset is a toydataset I generated that consists of 48 2d images. They vary across shape (triangle,square and circle), noise level (0%, 10%, 20% and 30%) and placements (1 per corner). Training data and test data each consists of 24 of these images. Varying over all shapes and offsets, the training data contains all images of the noise levels 0% and 20% and the test set contains images of noise levels 10% and 30%. Each image has dimensions 200×200.

The second dataset comes from (blank) and consists of 140 sagittal 3D MRIs of knees. They have several labels, but we will only focus on the health status. It's a boolean value where knees with a Kellgreen-Lawrence score of 1 and 2 is labeled healthy. Knees with Kellgreen-lawrence score of 3 and 4 are labeled unhealthy. The full dataset is unbalanced, having 99 healthy knees and 41 unhealthy. The training data consists of 68 MRIs, 47 healthy and 21 unhealthy. The test data consists of 72 MRIs 52 healthy and 20 unhealthy. Each MRI has dimensions $n \times 256 \times 256$, where n is the depth that ranges from 104 to 122.

The third dataset comes from Stanford University Medical Center and consists of a training set of 1130 knee MRI exams and a testset of 120 MRIs. The MRIs have 3 types of boolean labels: abnormal, acl and meniscus. The dataset includes sagittal, axial and coronal images for each knee. Only the sagittal scans have been used, in order to resemble the second dataset. A personal computer is used for this project, so to compute results in reasonable time, for each of the three tasks (abnormal, acl and meniscus) a trainingset of 100 MRIs and a testset of 100MRIs are extracted. All training and test datasets are balanced, having 50 positive cases and 50 negative cases each. Each MRI has dimensions $n \times 200 \times 200$, where n ranges from 17 to 47.

5 Method

Five methods were tested and they will each be described in this section. The methods are Haralick2d, Haralick3d, LBP2d, LBP3d and CNN2d. In this section, we'll describe the methodology used in prepossessing the images, calculating an AUC score, training the methods and finally how they are evaluated. Both 2d and 3d methods are used and a 2d slice of a 3d MRI is found by finding the depth of the MRI and slicing through the middle.

5.1 Prepossessing

The second dataset was cropped as the margin only contained zero values. The original shape was n * 256 * 256 and was cropped to n * 200 * 200. Here n denotes the depth axis
Images were then standardised by subtracting the mean, then dividing by 3 times the standard deviation. The images were then mapped to a pixel depth of 256.

The MRNet dataset wasn't cropped and the same standardisation as

5.2 Calculating the Probability Array and AUC

Given a method and a dataset, the AUC score with respect to some retrieval count K, is calculated the following way. The feature representation of each image in the dataset is calculated and saved and we'll denote this as our feature database (fd). Here fd[0] denotes the feature representation of the first image in our dataset and if l is the length of the dataset then fd[l-1] denotes the feature representation of the last image. After fd is calculated, we iterate through each entry fd[i] as our query image and use a distance measure to find the distance between fd[i] and each entry in fd (including fd[i]), and return the K+1 nearest indices of fd, minimizing this distance. The top ranking retrieved index is discarded as it's the same feature vector as the query vector. The remaining K indices are used to calculate the count of relevant results divided by K. The resulting ratio, represents a classification of the image with feature vector fd[i]. This process is repeated for each entry in fd and results in an array of size l, where each entry at index i is a decimal representing the classification of the image in the dataset at index i. We'll call this array the probability array, as each entry can be seen as a probability used to classify the image as either belonging to the positive or negative class. AUC score is then calculated using scikit-learns roc_auc_score function [4] with the image labels and the generated probability array.

[4]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

5.3 Haralick2d

The extraction method, Haralick2d, uses the Mahotas framework [5] to compute 13 features, given a 2d grey-level image. To select which of these features to use, sequential feature selection is used. This was implemented by a method that takes a number representing the maximum number of features that should be returned, a training set of 2d images and their associated labels. Beginning with an empty feature selection list [], each feature is used individually to calculate an AUC score for the entire training set. The feature maximising the AUC score is added to the list of selected features. The process is then repeated on the remaining features, each tried in combination with the already selected features. This process is repeated either until the maximum number of features is hit, or if adding an additional feature decreases the AUC score.
Euclidiean distance is used as distance measure.

5.4 Haralick3d

Haralick3d is fitted the same way as Haralick2d. In both Haralick2d and Haralick3d the maximum numbers of features to use is set to 5. This choice is both for effeciency, as Haralick3d is especially slow in calculating the AUC score, but also to avoid overfitting, as marginal gains might be indicative of dataset specific patterns rather than generalised patterns).
Euclidiean distance is used as distance measure.

5.5 LBP2d

Local Binary Patterns were computed using the Scikit-Image library [6]. First an LBP image is computed, then a histogram is created, expressing the occurrences of the integer values of the LBP operator at each pixel of the original image. Scikit-Image allows to specify either a default method or uniform, and uniform was chosen as described in the theory section. The histogram is then normalised. It's standard practice to either filter out zeros in the histogram, or apply smoothness. I applied smoothness, adding a value of 0.001 to each entry in the histogram and performing normalisation again.
LBP2d has two parameters, the radius of the circular neighbourhood surrounding a pixel center, and the amount of neighbours on this circle. As with Haralick2d, performance is measured by AUC score. Parameter combinations are iteratively tried until the best is found. To improve efficiency, I made the simplifying assumption that the neighbourhood count is proportional to the radius. Radius values between 1 and 15 are tested, and per radius value, neighbourhood count is calculated as multiples of the radius, in the range 1 to 10, ie from 1r to 10r.
J-divergence is used as distance measure.

[5] https://mahotas.readthedocs.io/en/latest/
[6] See: https://scikit-image.org/

5.6 LBP3d

LBP3d is similar to LBP2d. The LBP3d method is comrpised of three LBP2d objects, each calculating features in one of the 3 orthogonal planes. See figure x. The slice in each of these planes is the plane directly in the middle. Fx the chosen xy plane has a z-value which is half of the MRI depth. The LBP3d fitting is thus comprised of three LBP2d fittings. When features are extracted using LBP3d, the three histograms are computed from each of LBP3ds internal LBP methods and the histograms are concatenated.
J-divergence is used as distance measure.

5.7 CNN2d

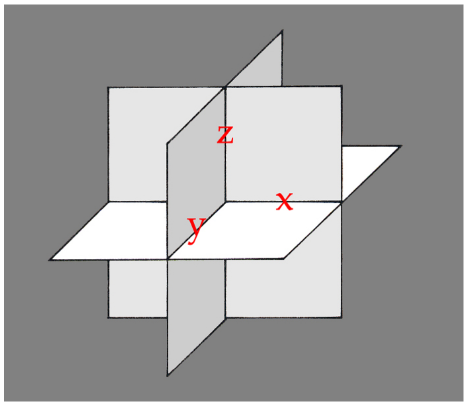


Figure 9: Two circular neighborhoods
http://facweb.cs.depaul.edu/sgrais/planes_and_space.htm

A convolutional neural network was implemented using Tensorflow [?]. Feature extraction was based on the approach in [15] where they after training a CNN, fed it with MR images and extracted the values from the last few fully connected layers, just before the classification, and used the extracted values from each of 3 fully connected layers as 3 separate feature vectors. They extracted from 3 layers to compare their performance, I however only extract from a single layer.
The architecture of the network is similar to what was used in [15]. It is a 2d method and takes as input $n*200*200$ which is n image slices of size $200*200$. Images are passed through 5 convolution layers of which the second, third and fifth also perform max pooling. All these layers use rectified linear unit. This is followed by 3 fully connected layers. The feature vector is extracted from the third fully connected layer. Softmax is applied to the final fully connected layer and is followed by a fully connected layer of size 2 and the final classification is made by taking argmax from this layer. The network is trained with softmax cross entropy as a cost function and Stochastic Gradient Descent as the optimizer. ML is fitted the usual way. With stochastic gradient descent as an optimizer, and a learningrate of e^{-3}, the optimizer is iteratively served 5 samples with associated labels, at a time, until it has been served the entire training set. Labels are one-hot encoded to a 2d vector with relevant labels encoded as [1,0] and non-relevant lables as [0,1].
A few differences from the CNN used in [15] should also be mentioned. 2 output neurons were used instead of 24 as the task in [15] had 24 classes. Stride length

- Here the output layer has 2 neurons instead of 24

Layer	filter size	filter count	output size	max pooing
1st Convolution layer	11	64	11 * 64	no
2nd Convolution layer	5	192	5 * 192	yes
3rd Convolution layer	5	384	5 * 384	yes
4th Convolution layer	3	256	3 * 256	no
5th Convolution layer	3	128	3 * 128	yes
1st fully connected layer	-	-	4096	no
2nd fully connected layer	-	-	4096	no
3rd fully connected layer	-	-	9	no
softmax layer	-	-	2	-
fully connected layer	-	-	2	no
argmax	-	-	-	-

5.8 Testing Stage

For each combination of dataset and method, evaluation is done the following way. The method is given training data to fit its' parameters. Then the method is passed a test dataset from which it calculates a probability array as described in the "Calculating AUC" subsection. The retrieval count for, K, is set to 5 for all tasks. Using the probability array and image labels, the ROC curve and AUC scores are plotted.

6 Results

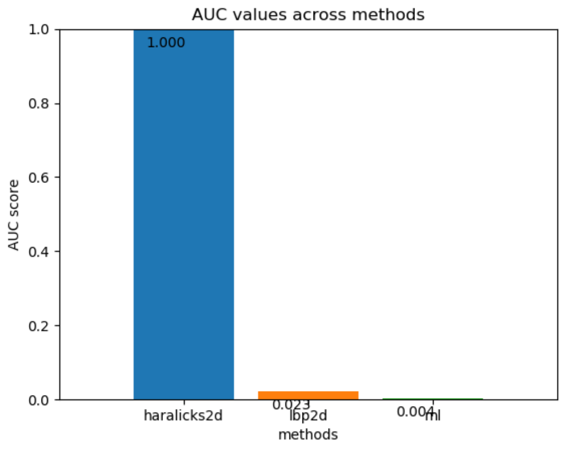
Below you see the performance of the five feature extraction methods on 5 different tasks. These tasks are:

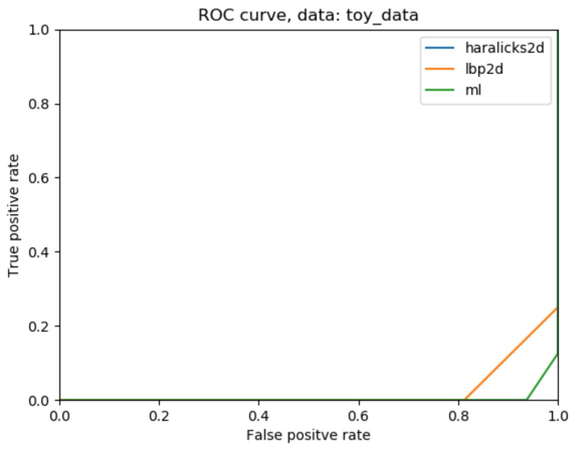
1. Retrieving images that are either triangles or not triangles, using the toy dataset
2. Retrieving images of correct health status, using Eriks dataset
3. Retrieving image of correct abnormal status, using the MRNet dataset
4. Retrieving image of correct meniscus status, using the MRNet dataset
5. Retrieving image of correct acl-tear status, using the MRNet dataset

6.1 Task 1

Here you see the results from the toy dataset on the task of retrieving triangles or non-triangles:

As these images are 2d, the 3d methods haven't been used. Of the 2d methods, Haralicks2d outperforms the others with an AUC score of 1.00. Both LBP2d and CNN2d show AUC scores near 0 which means they also show very high accuracy in discriminating the classes, they both however assign the wrong labels so they methods have high accuracy in assigning images from the positive class to the negative class and vice versa. As a classification method this is no issue as the labels can be reversed, but as a feature extraction method it's interesting that the two methods consistently find smaller distances across classes, than within classes.
Taking a look at the ROC curve, the Haralicks2s isn't visible as it with an AUC score of 1.0 hugs the upper left corner. The LBP2d and CNN3d curves look very similar. They are flat until around a fpr around 0.8-0.9 where they both spike with a consistent rate of change. The consistent rate of change is possibly because the test set is small. The curves suggest that the two classes are well seperated across most thresholds. Up until around 80% of the images can be separated without overlap for LB2d and around 95% for CNN2d.



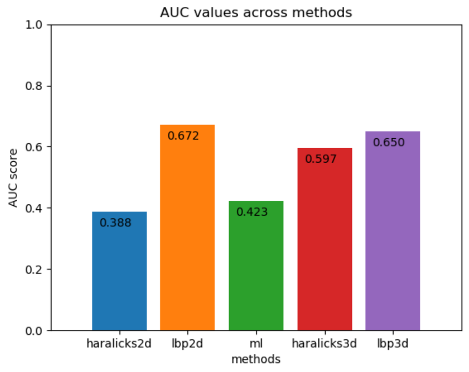


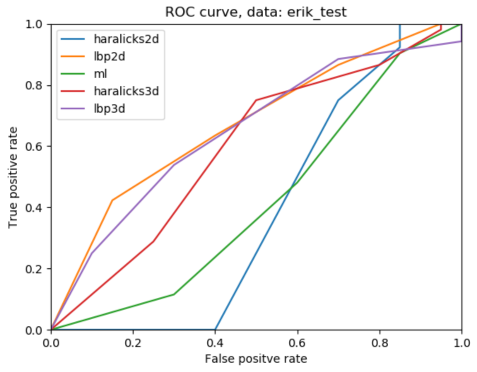
6.2 Task 2

Here you see the results from the erik dataset on the task of identifying healthy knees:

First taking a look at the AUC scores, the LBP2d shows best seperation of classes, closely followed by LBP3d and Haralicks2d. As with the toy data, the Haralicks2d and CNN2d are clos toegther, falling below 0,5. Overall non of the methods shows high accuracy in seperating the classes.

Looking at the ROC curves we see LBP2d rising the fastest, until it cuts just before a fpr of 0.2, showing a low non-chaning rate of change for the rest of the curve. LBP2d seems to be the only curve whose rate of change decreases so drastically. The Haralicks2d curve spikes after an fpr of 0.4. This suggests that the method can seperate 40% of the negative class accurately, but already at 60 fpr% there is a large overlap.

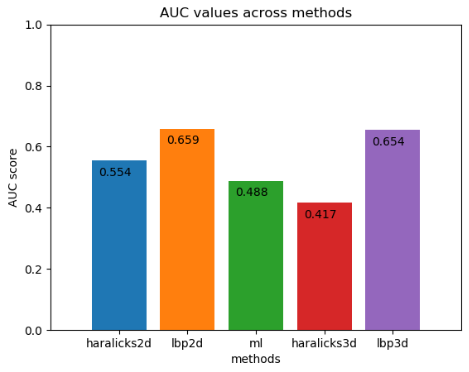


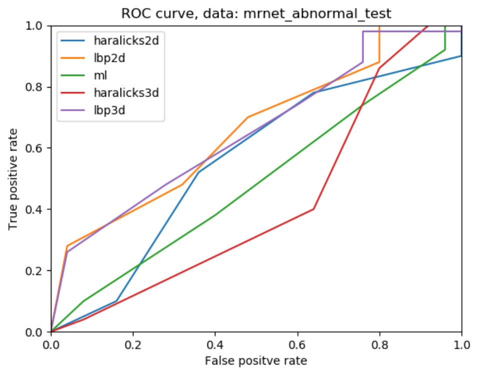


6.3 Task 3

Here you see the results from the mrnet dataset on the task of identifying abnormal knee scans:

Looking at the AUC scores, the LBP2d and LBP3d scores are similar slightly better than the rest. CNN2d, with an AUC score of almost 0.5, can't distinguish between the classes. Both Haralicks2d and Haralicks3d show low performance, but interestingly unlike in task 2, Haralicks2d has a score above 0.5 and Haralicks3d has a score below it.
Looking at the ROC curves, the LBP2d and LBP3d curves look very similar. As with task 2, there quickly is a drastic decrease in rate of change that stays consistent through the rest of the curve. Unlike task 2, the cut off is earlier, at a fpr around 0.05, indicating that the overlap increases as the methods try to separate more than 5% of the negative class. The CNN2d curve is nearly a diagonal line, showing that as the threshold is changed, false positives and true positives are added at the sme rate.
The Haralicks2d and Haralicks3d curves seems to have an inverse relationship, with haralick2d first increasing, then decreasing and vice versa for Haralick3d.

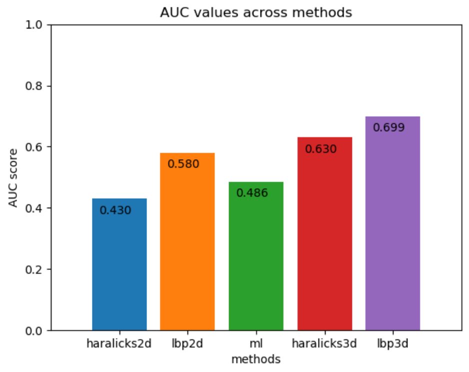


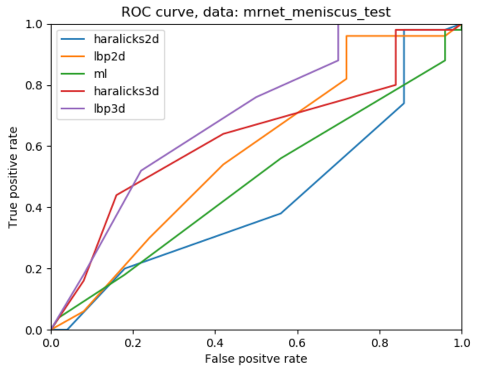


6.4 Task 4

Here you see the results from the mrnet dataset on the task of identifying a meniscus on knee scans:

Looking at the AUC scores, the LBP3d method shows highest accuracy with an AUC of 0.699. Again the CNN2d method seems no better than randomly classifying the images. The relationship between Haralick2d and Haralick3d is similar to in Task 2, where Haralick3d substantially outperforms Haralick2d.
Looking at the ROC curves, LBP3d and Haralicks3d look similar. Both decrease in rate of change around a fpr of 0.2. All in all the graph looks very similar to the one in Task 3, except that the position of Haralick2d and Haralick3d curves are interchanged.

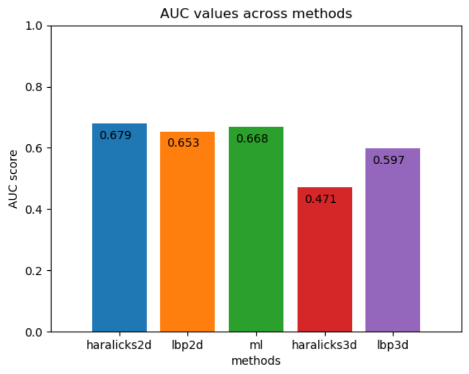


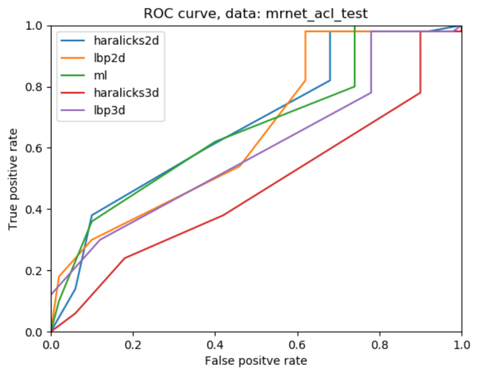


6.5 Task 5

Here you see the results from the mrnet dataset on the task of identifying acl tears on knee scans:

Looking at the AUC, Haralicks2d, LBP2d and CNN2d all show a similar performance. Unlike the other tasks, the CNN2d is both above 0.5, and one of the leading methods in accuracy. As has been seen in the other tasks, the inverse relationship between Haralicks2d and Haralicks3d remains. We see that Haralicks2d shows the best performance and Haralick3d, the worst.
Looking at the ROC curves, unlike most of the tasks, these curves are very homogenous. All break around fpr of 0.1-0.2 and share the same rate of change for the rest of the course. This is with LBP2d as an exception, as it has a spike around an fpr of 0.5. This indicates that we should be more sceptial of its' AUC score, as the spike inflates the score even though it shows a far higher fpr than Haralicks2d and CNN2d at tpr values before the spike.





References

[1] Madugunki, M., Bormane, D. S., Bhadoria, S., & Dethe, C. G. (2011). Comparison of different CBIR techniques. In 2011 3rd International Conference on Electronics Computer Technology. IEEE.

[2] Li, Z., Zhang, X., MÃ¼ller, H., & Zhang, S. (2018). Large-scale retrieval for medical image analytics: A comprehensive review. Medical Image Analysis, 43, 66-84.

[3] Haralick, Robert and Shanmugam, K and Dinstein, Ih. (1973). Textural Features for Image Classification. IEEE Trans Syst Man Cybern, 3, 610-621.

[4] Mryka Hall-Beyer. (2017). GLCM Texture: A Tutorial v. 3.0 March 2017. University of Calgary.

[5] Ojala, T., Pietikainen, M., & Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7), 971-987.

[6] Depeursinge, A., Fageot, J. (2017). Biomedical Texture Operators and Aggregation Functions. In Biomedical Texture Analysis (pp. 55-94). Elsevier.

[7] Pandit, S., & Gupta, S. (2011). A Comparative Study on Distance Measuring Approaches for Clustering. International Journal of Research in Computer Science, 2(1), 29-31.

[8] Han, Jiaweim (2008), Kullback-Leibler Divergence. lecture note from University of Illinois, Urbana Champaign in the course Introduction to Data Warehousing and Data Mining. url: http://hanj.cs.illinois.edu/cs412/bk3/KL-divergence.pdf. Last visisted 23. May 2019.

[9] Shibuya, Naoki, Demystiying Entropy. (2018). Towards DataScience. url: https://towardsdatascience.com/demystifying-entropy-f2c3221e2550. Last visisted 23. May 2019.

[10] Shibuya, Naoki, Demystiying Cross-Entropy. (2018). Towards DataScience. url: https://towardsdatascience.com/demystifying-cross-entropy-e80e3ad54a8. Last visisted 23. May 2019.

[11] Shibuya, Naoki, Demystiying KL-Divergence. (2018). Towards DataScience. url: https://towardsdatascience.com/demystifying-kl-divergence-7ebe4317ee68. Last visisted 23. May 2019.

[12] https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[13] Albawi, S., Mohammed, T. A., Al-Zawi, S. (2017). Understanding of a convolutional neural network. In 2017 International Conference on Engineering and Technology (ICET). IEEE.

[14] https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[15] Qayyum, A., Anwar, S. M., Awais, M., Majid, M. (2017). Medical image retrieval using deep convolutional neural network. Neurocomputing, 266, 8-20.

[16] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 60(2), 91-110