Using Machine Learning Algorithms to Detect Brain Tumours

Chris Karvelas and Ong Jing Xiang

MATH 3490: Optimization

Dr. Shaun H. Lui

April 27, 2023

1. Introduction

1.1. Background

In the medical field, the problem of detecting cancer is a major area of focus. According to Ritchie, et al. (2019), cancer is the second leading cause of death around the world, killing over 10 million people in 2019. The Oxford dictionary defines a tumour as "a mass of cells growing in or on a part of the body where they should not, usually causing medical problems" ("Tumour", n.d.). A tumour can be either benign (not cancerous), or malignant (cancerous). According to Cancer Research UK, in the United Kingdom, from 2010-11, the ten-year survival rate for all cancer overall was 50% and has doubled in the last 40 years. Although this drastic increase in survival rates is a very positive development, there is still a long way to go in the fighting cancer. Even more troubling is the fact that there is major variation in the survival rates of different cancers, ranging from a 10-year survival rate of 98% for testicular cancer, to just 1% for pancreatic cancer in the UK ("Cancer Survival Rates", n.d.). One particularly dangerous type of cancer is brain cancer with a 10-year survival rate of just 11% in the UK from 2013-2017. Additionally, brain cancer is practically unpreventable, with just 3% of cases being considered preventable ("Brain, other CNS and intracranial tumours statistics", n.d.). As a result, the best tool for fighting these cancers is early detection.

According to the American Cancer Society, the two most common ways of detecting brain tumours are magnetic resonance imaging (MRI) and computed tomography (CT) scans. Both scans are very effective at showing brain tumours if a patient has one, but of the two methods, MRI scans are preferred, as they include more detail than a CT scan ("Tests for Brain and Spinal Cord Tumors in Adults", 2020). According to Johns Hopkins University, although brain tumours can be non-cancerous, all brain tumours are dangerous, as they put pressure on the brain and can potentially spread to other regions. Additionally, in some instances, benign brain tumours can become malignant, therefore, any brain tumour should be removed as soon as possible. In the United States, brain and nervous system tumours affect around 100 000 people. Furthermore, brain tumours are the most common type of solid tumour in children, impacting around 5 000 children each year ("Brain Tumors and Brain Cancer", n.d.). Given this large presence of brain tumours and the low survival rate, being able to detect tumours early on, before they have the chance to spread is a very important task.

With the advent of computers and machine learning, the ability to quickly process large amounts of data has become a very useful tool in many industries. One industry that can benefit from such advancements is the medical industry. Being able to quickly and accurately classify large amounts of diagnostic data will greatly benefit the medical profession. This improvement would be clear in the field of Oncology, where cancer detection is a race against time so that the cancer can be treated before it becomes too late. This project therefore aims to use a variety of machine learning algorithms and data pre-processing techniques to detect cancer from MRI scans of the brain. Throughout this report, we hope to find out which models and pre-processing techniques lead to the best outcomes that will most accurately classify scans as positive, or negative, and do so in a reasonable amount of time.

1.2 Data

In the report, all algorithms were run on a series of brain scans taken from a dataset on Kaggle. The dataset includes 3060 scans of brains, with 1500 positive scans (with a tumour), 1500 negative scans (without a tumour), and 60 unclassified scans, which are meant to be predicted. Overall, the tumours appear as lighter spots of varying size and location on the brain, however, they are not always obvious. The hue of the scans varies widely, so there is quite a bit of variation in what a tumour looks like. Below are a few examples of positive scans:

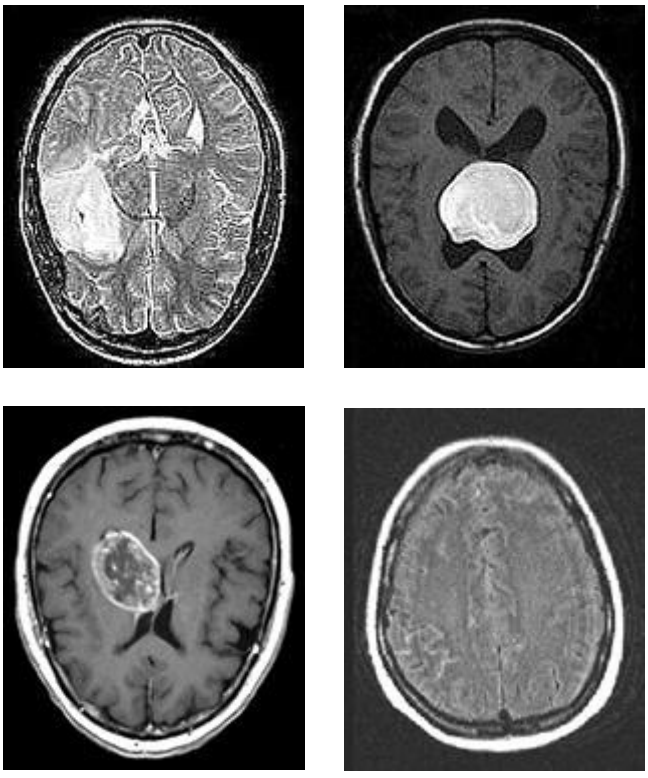

Figure 1: Four different examples of positive brain scans. Notice the stark different in hue of the top two images. It is also clear that the tumour is lighter than the brain around them in the top two images, with one on the left side middle in the top left image and one in the middle of the top right image. On the bottom, however, the tumour in the left image is quite a lot darker. On the bottom right the tumour is very hard to see.

Due to some unclear scans, there are many challenges we face when attempting to classify such images, although many are clear, there are some that are very difficult, or nearly impossible to conclusively classify.

Like the positive scans, the negative scans vary widely. The brightness levels of the images vary and some of the images also have spots that appear to be similar to how a tumour

appears. Below are pictured some examples of negative brain scans that illustrate the range of image styles present in the data.

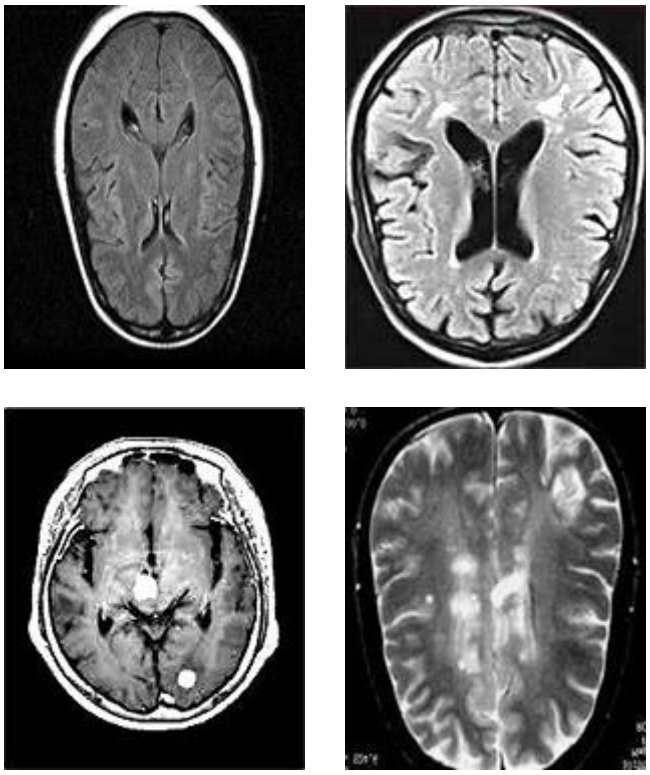

Figure 2: Four brain scans that do not contain tumours. Notice the top left image is significantly darker than the top right image. The bottom left image has a few light spots that seem to somewhat resemble how tumours appear. There are also some lighter spots on the bottom right image as well.

Additionally, some of the brains in the scans look rather different, with some parts of the brain being clear in some scans but blurry or invisible in others. There is a lot of variation in the scans so if our methods can overcome these obstacles and produce accurate results, then it will indicate that the methods are robust and able to deal with such differences in scans.

This report primarily aims to test and measure the accuracy of the models. Since the 60 prediction images were not assigned a label, we are not going to include them in our analysis. Some images are hard to classify to the untrained eye, so we do not want to misclassify an image by assigning labels based on what we think. We will instead test the models on the pre-classified data so we can get a proper accuracy score, knowing that the target value is correct.

2. Explanation of Methods

2.1. Support Vector Machine (SVM)

A support vector machine is a supervised Machine Learning algorithm. SVMs are commonly used for solving classification problems (Radhika, 2020). For binary classification problems like this one, it divides the data into two groups and determines the ideal hyperplane which divides the data points into two components and attempts to maximize the margin. There are two different types of margins classification, hard margin, and soft margin (Karimi, 2022).

Given m data points $x_j \in R^N$, each of which are classified into one of two groups:

$$y_j \in \{-1,1\},(j = 1, \ldots., m).$$

The hyperplane function is:

$$f(x) = w^T(x) + b = 0$$

Where the vector *w* is the orientation of a hyperplane plane, and the scalar *b* is the intercept.

Given two separable sets, we want to calculate the hyperplane so that it bisects the distance between these two sets, class 1, and class 2. Any value on one side of the hyperplane will be classified as belonging to class 1 and any point on the other side will be classified as belonging to class 2. For example, if both classes are convex hulls, we can find the closest points in one set to the other set and construct the plane that divides these two points (Bennett & Campbell, 2000). These points that are closest to the hyperplane are called support vectors (Gandhi, 2018).

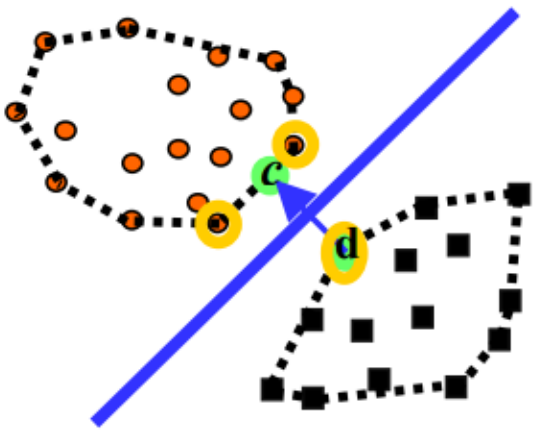

Figure 3: Best plane divides closest points in the convex hulls as part of hard margin classification

The formula, according to Bennett & Campbell (2000), to find the closest points in the two convex hulls is as follows:

$$\min_a \frac{1}{2} \| c - d \|$$

Where:

$$c = \sum_{y_j \in Class1} a_j \, x_j \text{ and } d = \sum_{y_j \in Class2} a_j \, x_j$$

$$\text{s.t} \sum_{y_j \in Class1} a_j = 1, \ \sum_{y_j \in Class2} a_j = 1$$

$$a_j \geq 0 \qquad j = 1, \ldots, m$$

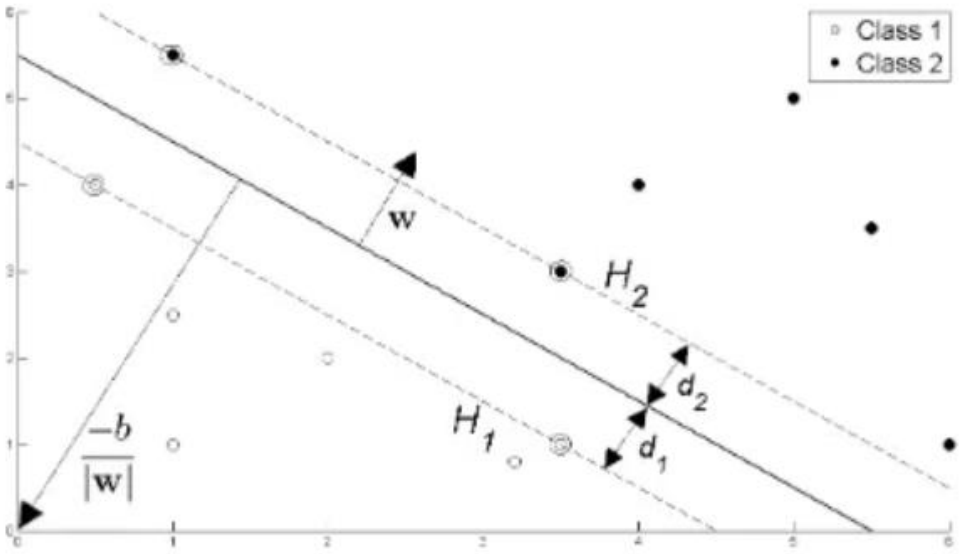

Figure 4: A hyperplane that maximizes the margin.

Additionally, we need to maximize the margin between two parallel supporting planes (the dotted lines in Figure 4). In hard margin classification, a plane supports a class if all points in that class are on one side of that plane. From Figure 4, there are two hyperplanes, H_1 and H_2.

The function of H_1 is:

$$w * x_j = b - 1$$

The function of H_2 is:

$$w * x_j = b + 1$$

The distance between H_1 and H_2 is the margin is as follows (MLMATH.io, 2019):

$$M = \frac{1 - b}{|w|} - \frac{-1 - b}{|w|}$$

Which simplifies to:

$$M = \frac{2}{|w|}$$

Since,

$$max\,\frac{1}{\| w \|}$$

Which is equal to:

$$min \| w \|$$

Therefore, if we want to maximize the margin, we solve this minimization problem:

$$\min_{w,b} \frac{1}{2} \| w \|^2$$

s.t

$$w * x_j \geq b + 1, y_j \in Class1$$

$$w * x_j \leq b - 1, y_j \in Class2$$

The constraints can be simplified to:

$$y_j(w * x_j - b) \geq 1.$$

In this project, we will use soft margin classification, which is a variant of SVM. Soft margin classification is useful for when you cannot perfectly separate the data with a hyperplane, so it allows for some misclassifications to happen. However, we still need to minimize the misclassification error (Karimi 2022). We also need some slack variables to give the margin wiggle room in each dimension (Brownlee, 2016).

Therefore, the formula of the soft margin is:

$$min\frac{\| w \|^2}{2} + C\sum_{i=1}^{n} \zeta_i$$

$$\text{s.t.} \qquad y_i(w^T x_i + b) \geq 1 - zeta_i \qquad \zeta \geq 0$$

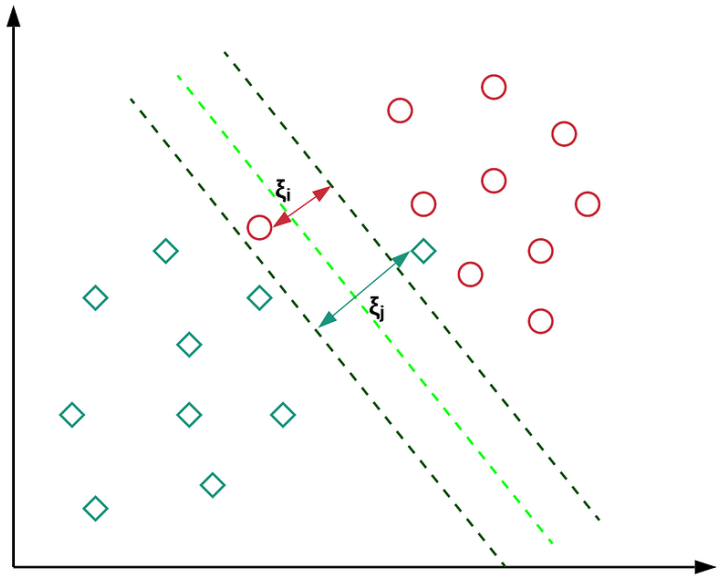

Figure 5: A simple example of soft margin classification.

The training of SVMs are slow, but we can use Sequential Minimal Optimization (SMO) to quickly solve the SVM quadratic programming problem.

This is the formula of SMO:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j K(x_i, x_j) \alpha_i \alpha_j,$$

subject to:

$$0 \leq \alpha_i \leq C, \quad \text{for } i = 1, 2, \ldots, n,$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

Where $K(x_i, x_j)$ is the kernel function, C is an SVM hyperparameter, and the α's are Lagrange multipliers. It simplifies the problem by solving the dual problem as opposed to the primal problem.

Therefore, if the data is separable, we use hard margin classification, otherwise, we use soft margin classification (Brownlee, 2016).

2.2. K-Nearest Neighbours (KNN)

The KNN machine learning algorithm is simple yet effective. It is a supervised and non-parametric learning algorithm. KNN is commonly used for classification problems to classify data based on the closest neighbouring data points in a certain area (Taunk et al., 2019). The distance between the new data point and the other data points needs to be calculated. There are several methods to calculate the distance. First, the Euclidean distance between two points p and q in n-space is give calculated by:

$$D(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

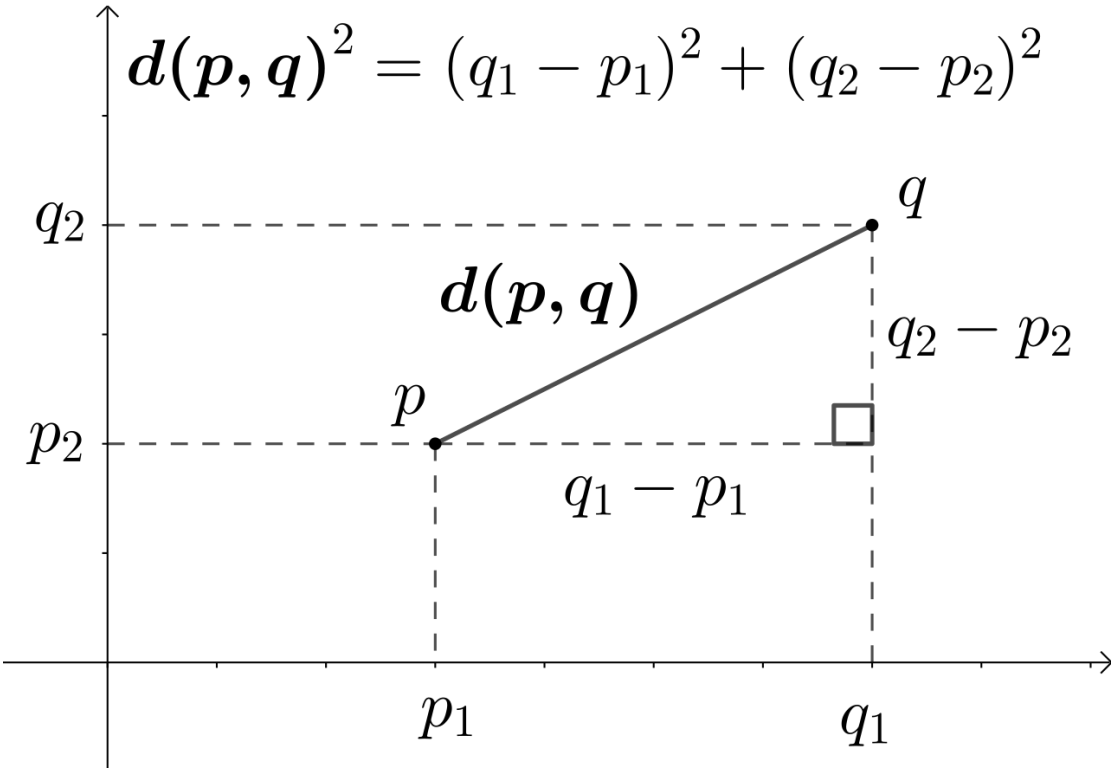

Figure 6: A diagram of Euclidean distance, which shows the straight-line distance between two points.

Another type of distance is Manhattan distance, which is calculated by:

$$D(p,q) = |p_1 - q_1| + |p_2 - q_2| + \cdots + |p_n - q_n|$$

It is calculated as the sum of absolute differences between the new data point and the other point.

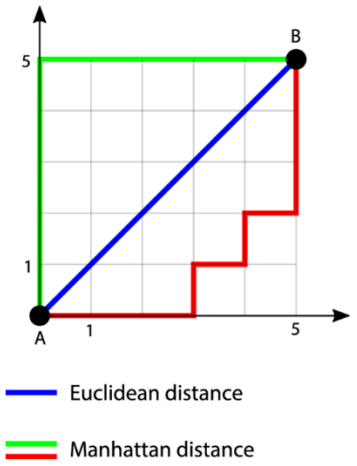

Figure 7: A visual comparison of what Euclidean and Manhattan distances represent when finding the distance between points A and B.

Another form of distance is Minkowski distance, which is given by the following equation:

$$D(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^k\right)^{\frac{1}{k}}$$

This distance is a generalization of the Euclidean and Manhattan distance metrics. The norm order, k allows for the creation of the other distance metrics. When $k = 1$, this distance measure calculates the Manhattan distance, and when $k = 2$, it calculates the Euclidean distance ("K-Nearest Neighbors Algorithm", n.d).

Another important aspect of KNN is choosing the value of k for the number of closest training data points to consider. To avoid either overfitting or underfitting, several values of k should be considered. Larger values of k may result in a high bias and low variance, whereas lower values of k may result in high variance and low bias ("K-Nearest Neighbors Algorithm", n.d). We can run the classifier multiple times with different values of k to determine which value produces the best results, but it takes $O(N^2)$ time to determine the values of k. (Guo et al., 2004). The new data point is classified into the class that most of its k nearest neighbors belong to ("K-Nearest Neighbors Algorithm", n.d). It is, therefore, good to use an odd number for k in a binary classification setting to avoid ties.

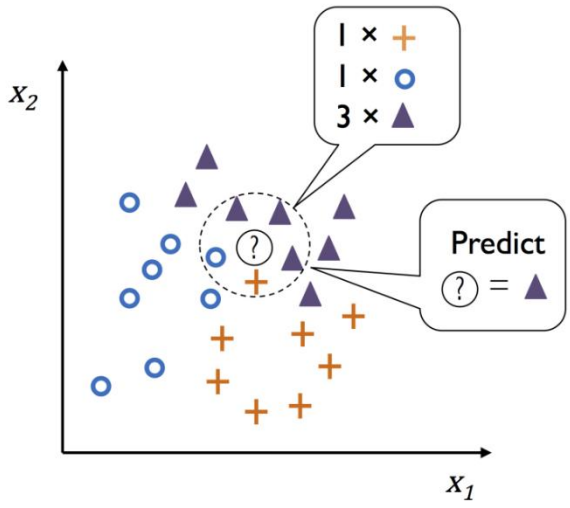

Figure 8: An example of KNN for a 3-class problem with $k = 5$.

2.3. Principal Component Analysis (PCA)

In large data sets, we use principal component analysis to reduce the dimension of the data in an interpretable way. PCA transforms a large data set of variables into a smaller one, while at the same time minimizing loss of information. Therefore, PCA removes unimportant information and noise from the dataset (Jaadi, 2023).

In PCA, we need to standardize the data to ensure that each variable is given equal importance (Jaadi, 2023). For example, variables with high variance will be more important than the variables with low variance. Therefore, we need to transform the data to comparable scales, using the following formula:

$$z = \frac{x - \mu}{\sigma}$$

Where z is the standardized value of the variable, x is the value, μ is the mean, and σ is the standard deviation of the variable (Jaadi, 2023).

After standardizing the data, we can compute the covariance matrix, which helps us understand how the variables are linearly related. If a variable is highly correlated, it likely contains unimportant information.

The covariance matrix is a $p \times p$ symmetric matrix, where p is the number of variables in the data set. Each element in the matrix represents the covariance between two variables.

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Figure 9: The covariance matrix of the data set with 3 variables.

The following formula calculates the covariance of any two variables, *X*, and *Y*:

$$cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}\big(x_i - E(X)\big)\big(y_i - E(Y)\big)$$

Where *n* is the number of data points and *E(·)* is the expected value of a variable.

After computing the covariance matrix, we compute its eigenvalues and eigenvectors to identify the principal components of the data. The principal components are the linear combinations of the original variables. The first principal component is the linear combination with the most variance in the data set, the second principal component is the linear combination with the second most variance and is perpendicular to the first principal component, and so on. We calculate *p* principal components (Jaadi, 2023). The first principal component corresponds to the eigenvector with highest eigenvalue. From here, we compute the percentage of explained variance (information) of each component. Finally, we can specify what amount of overall explained variance, v we want and then keep the first k principal components that account for v% of total variance, discarding the rest (Jaadi, 2023).

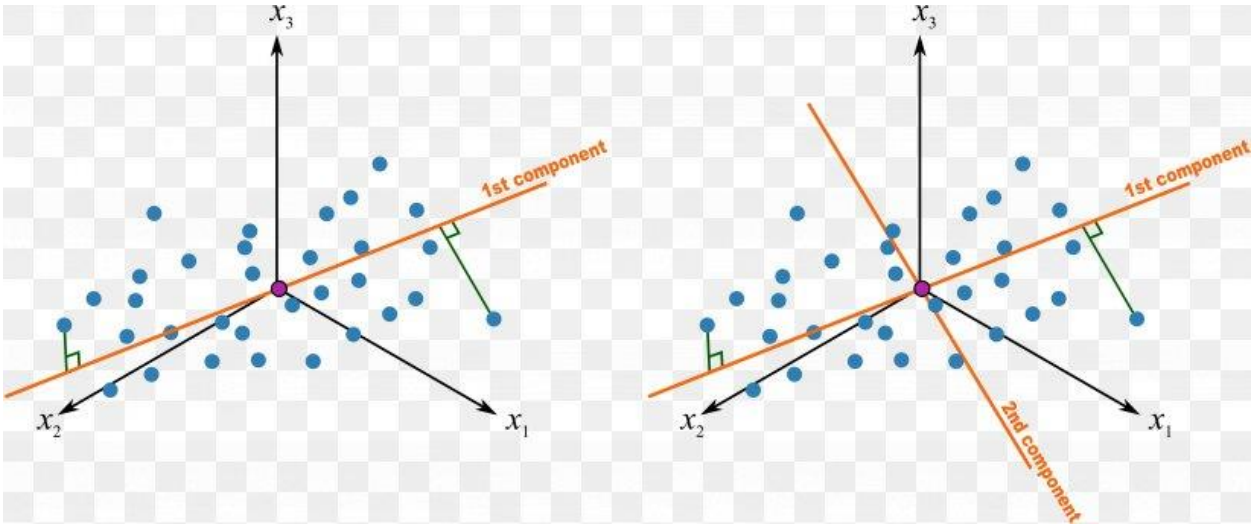

Figure 10: An example of PCA with the first two principal components.

The formula to calculate the variance of the i^{th} principal component of matrix *Y*, is as follows:

$$var(Y_i) = \sum_{k=1}^{p} \sum_{l=1}^{p} e_{1k}\, e_{1l} \sigma_{kl}$$

Where the *e*'s are the coefficients of the linear combination of the i^{th} eigenvector's elements ("Principal Component Analysis (PCA) Procedure", n.d).

A major part of PCA is singular value decomposition (SVD). SVD states that any matrix, *A* can be factorized as:

$$A = USV^T$$

Where *U* and *V* are orthogonal matrices with orthonormal eigenvectors chosen from AA^T and A^TA respectively and S is a diagonal matrix with *r* elements equal to the root of the positive eigenvalues of AA^T or A^TA. The diagonal elements are composed of singular values. (Hui, 2019). We can ignore the vectors corresponding to small singular values (Lui, 2023). An example of SVD performed on an image in the dataset is shown below:

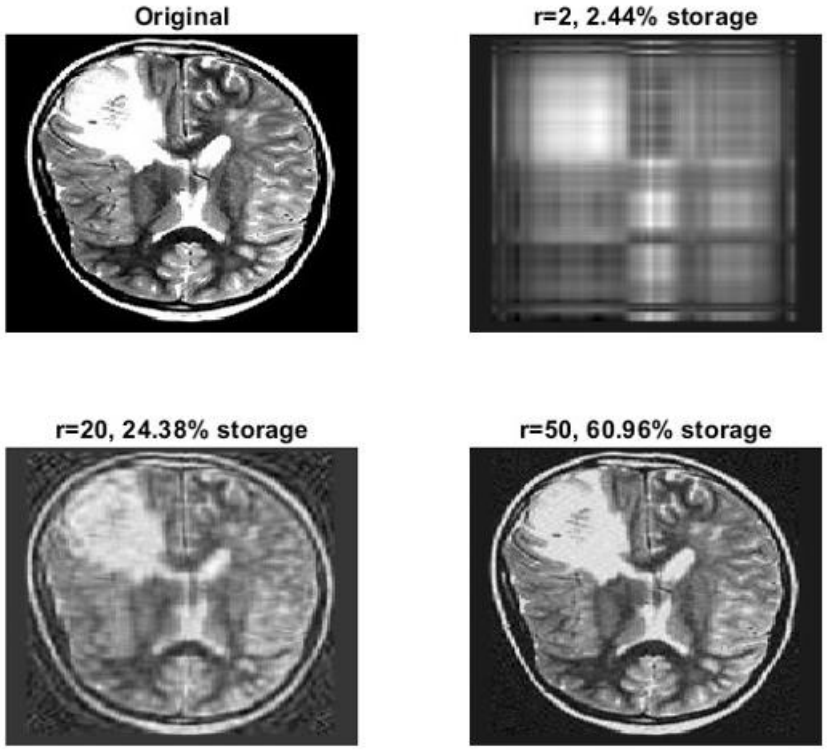

Figure 11: SVD on one of the brain scans demonstrating what happens as r increases and documenting the percentage of storage taken compared to the original image.

2.4. Non-Negative Matrix Factorization (NNMF)

The purpose of non-negative matrix factorization is to reduce the dimensionality of the data. It is usually used in image processing. NNMF decomposes a non-negative matrix V into two lower-rank non-negative matrices W and H, where:

$$V \approx WH$$

The matrix W contains the basis vectors and H contains the associated coefficients (weights).

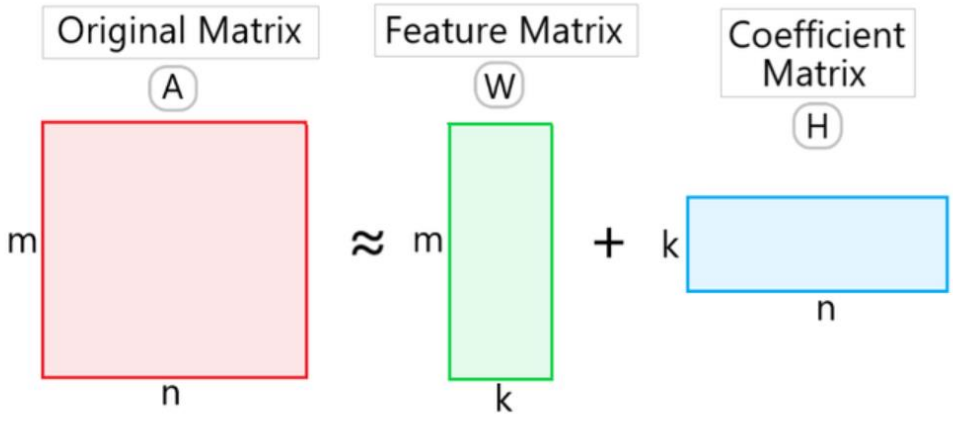

Figure 12: The decomposition of A into matrices W and H.

Therefore, we obtain two lower dimension matrices than the original matrix.

The cost function of the NNMF is as follows:

$$\| V - WH \|^2 = \sum_{ij} \left(V_{ij} - WH_{ij}\right)^2$$

The function will equal zero if and only if $V = WH$.

To minimize the cost, we need to minimize $\| V - WH \|^2$ with respect to W and H, subject to the constraints $W, H \geq 0$ (Lee & Seung, n.d.).

2.5. Convolutional Neural Network (CNN)

Convolutional neural networks are capable of learning spatial hierarchies of features automatically and adaptively. Convolutional layers, pooling layers, and fully connected layers make up CNNs (Yamashita et al., 2018). When these layers are stacked, a CNN architecture has been formed. In Figure 13, a simplified CNN architecture is shown. Applications of CNNs include image classification, facial recognition, autonomous driving, and much more.

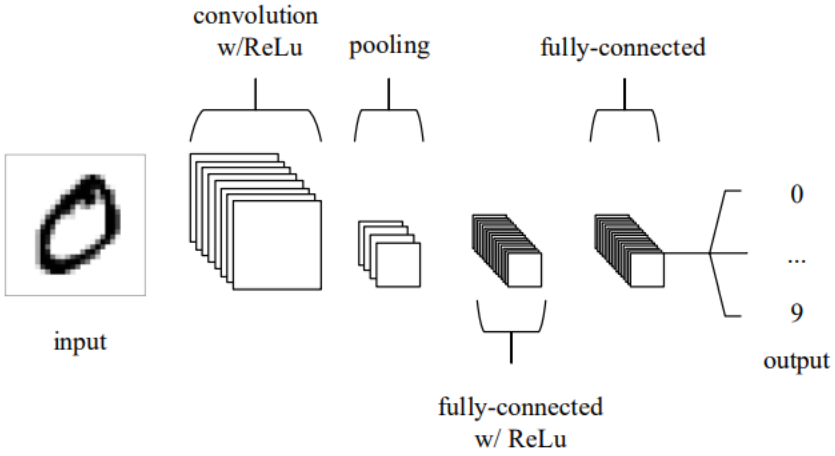

Figure 13: A simple CNN architecture

For CNNs to function, the convolutional layer is essential. We need to use a kernel in this layer. A kernel involves passing a small matrix of numbers across our picture to change it based on the values from the filter. This procedure is also known as filtering. The following formula is used to determine subsequent feature map values:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I\,(m,n)K(i-m, j-n)$$

The input image is denoted by *I* and our kernel by *K*. where *m* and *n* are the indices of rows and columns of the resulting matrix respectively and *S(i,j)* is the feature map values of the i^{th} row and j^{th} column.

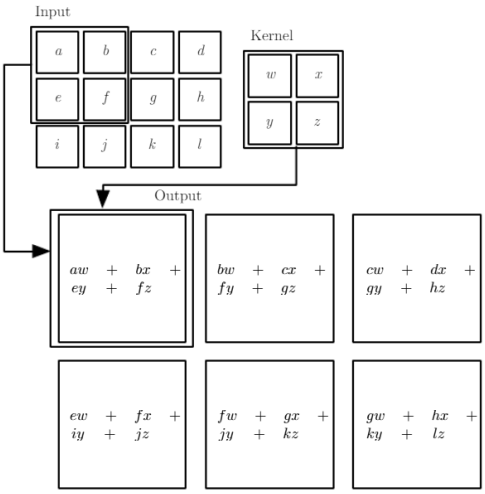

Figure 14: An example of 2-D convolution.

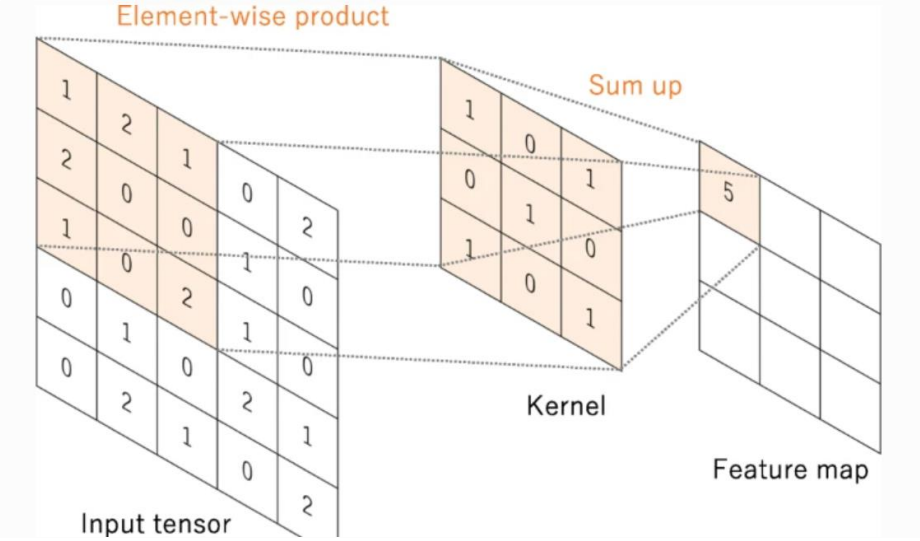

Figure 15: An example of convolution operation with a kernel size of 3 × 3.

It is common to add a pooling layer after a convolution layer. Pooling layers help reduce the dimensions of the data, lowering the model's computational complexity and parameter count. Pooling layers are similar to the filter and can be applied to feature maps. For images. the size of the filter is usually 2x2 pixels and is typically applied with a stride of 2 pixels. The stride argument specifies the step size of the training function as it scans the input ("Create Simple Deep Learning Neural Network for Classification", n.d.). The stride should always be smaller than the feature map's size. In the end, the pooling layer will divide the size of each feature map by two.

There are two popular forms of pooling operation: max pooling and average pooling. From the input feature maps, max pooling selects patches, outputs the largest value in each patch, and discards all other values. This results in a two-fold downsampling of feature map's in-plane dimension. However, the depth dimension of feature maps does not change.

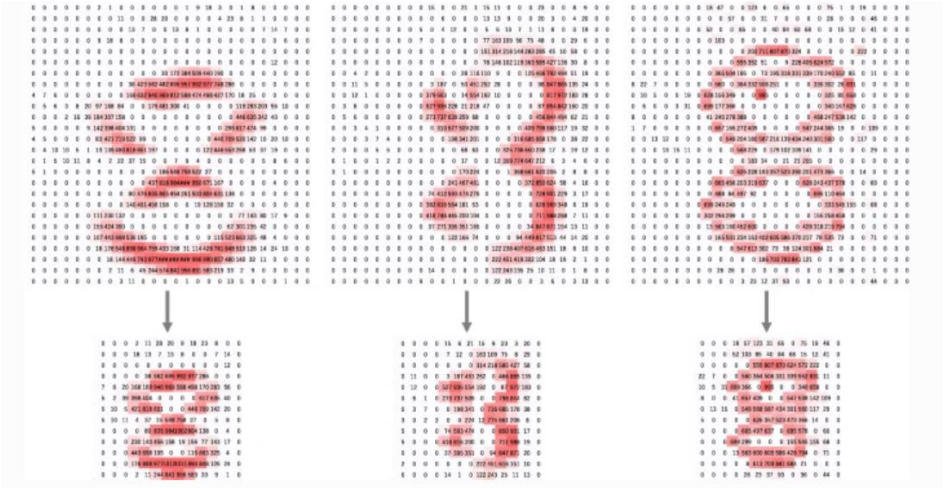

Figure 16: Examples of the max pooling operation on the same images. Notice the size reduction in each case.

From the input feature maps, average pooling selects patches and outputs the average of all values in each patch.

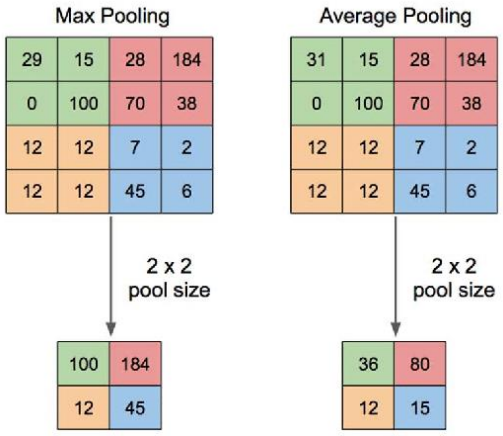

Figure 17: Examples of the pooling operation.

After using convolutional and pooling layers, the features are mapped by a subset of fully connected layers to the final outputs of the network. Each fully connected layer is followed by a nonlinear function. The purpose of a fully connected layer is classification in neural networks.

Neural networks are a set of dependent non-linear functions. Each individual function consists of a neuron. The neuron in a fully connected layer transforms the input vector linearly using a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function, f (Unzueta, 2018).

$$y_{jk}(x) = f\left(\sum_{i=1}^{n_H} w_{jk}\, x_i + w_{j0}\right)$$

The weights matrix is denoted by w and the function, f is a non-linear activation function that applies a non-linear transformation. Where x is the input vector and y is the output vector.

3. Explanation of Methods

3.1. Data Pre-Processing

For any of the models to be trained, the images must all be the same size. Unfortunately, the image sizes varied and in many cases some images were several times larger than others. Initially, we cropped the images to be a certain size, however, there were many images in which parts of the brain were cut off, which made them harder to classify. After a closer examination of the images, we realized that the aspect ratio of each image was nearly identical. As a result, we cropped the original dataset again and maintained the aspect ratio, which fixed the problem of the brains being cut off and led to each image having the same size, 181x150 pixels.

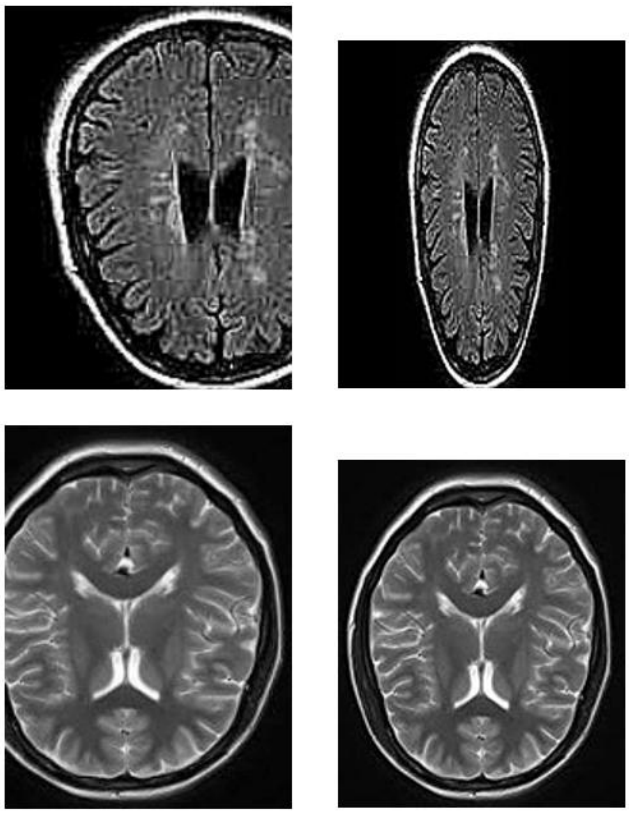

Figure 18: Two examples of how the images looked in the first cropping (left side) and the second cropping (right side). The top left image has a substantial amount of the brain cut off, which, although it was a negative scan, could have potentially hidden a small tumour. Unfortunately, the second cropping, although displaying the whole brain, distorted the image by stretching it vertically. The bottom row shows a much nicer cropping with no distortion; most images ended up non-distorted like on the bottom right.

In this report, we chose to use SVM and KNN to test, as they are commonly used algorithms for classification, and they are easy and relatively quick to train.

The image importing was done differently for the SVM and KNN models than the CNN. For the SVM and KNN models, the images were loaded into MATLAB as a 3-dimensional matrix, 2 dimensions for the height and width, and a third for each image. Since the name of each image was indexed from 0-1499, starting with "y" for positive screens and "no" for negative screens, all possible index values were permuted. The training data consisted of images whose indices were the first 1200 permuted values. For example, if 173 was the first number in the permutation, the images "y173" and "no173" would be the first 2 images loaded into the training set. The images whose indices were in the last 300 permuted numbers were loaded into the test set. This resulted in 2400 training images and 600 testing images, an 80-20 train-test split, each with an equal number of both classes. A random seed was also set for reproducibility. When a positive image was loaded in, the corresponding target value was set to 1 and when a negative image was loaded in, the target was set to 0.

Once the images were loaded, the data was reshaped to become a 2-dimensional matrix, where each individual pixel became a column, resulting in a 2400 x 27 150 matrix training set and a 600 x 27 150 test set.

CNNs on the other hand handle images differently and thus had a different loading process. The images were loaded using an *imageDataStore* and were classified based on the name of the folder they were stored in. Finally, 1200 images from each were taken as training data and the other 300 from each were taken as validation data, which acted the same as the test data from the KNN and SVM models.

3.2 Results

3.2.1 Test 1: No Extra Pre-Processing

The first test that was performed was on the original data, without any extra pre-processing, besides what was required to allow it to be used to train the models. The accuracy and time to train and test each model is recorded below:

Model	Training Time	Accuracy
SVM	27.9 sec	96.2%
KNN	13.1 sec	97.0%

Note: All the SVM models from the tests had the same optimal hyperparameters. They had linear kernels (linear hyperplanes) and were solved using SMO. KNN's parameters differed slightly, but all KNNs were optimized when they used $k = 1$ nearest neighbour and either Minkowski or Euclidean distance.

From the above results, we see that the two machine learning models are very effective at detecting tumours, each achieving over 96% accuracy on the test set, which is a good start. Since this test was done without any special pre-processing, these results will be used as a baseline to compare the results of the other tests to. We hope to improve upon these results with additional pre-processing.

Furthermore, we observe that the models performed similarly, but that the KNN model took less than half as long to train and predict. As a result, the KNN model appears to be much more efficient.

3.2.2 Test 2: Principal Component Analysis (PCA)

The first type of additional pre-processing performed was PCA. We reused the training and test datasets from the first example and performed PCA on them. We tried several levels of explained variance and recorded the results from each.

Below are the results from several levels of explained variance:

Explained Variance	Number of Components	Training Time SVM	Training Time KNN	Accuracy SVM	Accuracy KNN
50%	20	18.4 sec	0.2 sec	41.7%	97.7%
75%	116	19.4 sec	0.3 sec	69.2%	96.3%
85%	229	19.3 sec	0.3 sec	66.2%	95.8%
95%	497	16.5 sec	0.7 sec	93.2%	96.3%
97%	624	13.9 sec	0.3 sec	95.7%	96.5%

Note: Calculating and extracting the PCA components took on average about 25 seconds, without any significant difference at the various levels.

Overall, it is hard to see any improvement with PCA in terms of both time elapsed and accuracy. Without PCA, KNN took around 13 seconds and SVM around 27 seconds. Although both models took less time to train with PCA, if you factor in the time taken to perform PCA on the dataset, each method took longer to train.

In terms of accuracy, there was a wide variety of results. Overall, keeping the components that explain 97% of the variance in the dataset gave in the best results for the SVM model and did so with the shortest training time. The result of 95.7% accuracy was slightly lower than without PCA, which was 96.2%. Curiously enough, keeping a low number of components led to the best result for the KNN model. Keeping 50% of explained variance, which resulted in only 20 components, led to 97.7% accuracy, an improvement over the first test.

3.2.3 Test 3: Non-Negative Matrix Factorization (NNMF):

The second pre-processing method we used was NNMF. The training and test data from the previous two tests were reused. First, we stacked the training and test data to apply NNMF to it, before multiplying the factors back together and re-splitting the data, preserving order; the first 2400 rows were still the training data and the last 600 were still the test data. Since the data consisted of pixel values, everything was non-negative, so we can use this method. The results for several numbers of features and elapsed time are detailed below:

Number of Features	Elapsed Time NNMF	Training Time SVM	Training Time KNN	Accuracy SVM	Accuracy KNN
50	1020.8 sec	65.9 sec	0.2 sec	55.8%	98.0%
25	313.6 sec	64.5 sec	0.2 sec	62.2%	97.8%
10	123.7 sec	54.4 sec	0.6 sec	36.8%	97.8%
5	34.2 sec	25.5 sec	0.2 sec	46.0%	98.0%

Non-negative matrix factorization scaled incredibly inefficiently, with the time required increasing exponentially as we added features. Furthermore, it performed horribly with the SVM model, scoring much lower than any of the other tests in that regard. It did, however, performs quite well with the KNN model, scoring higher than any of the other tests so far, reaching 98% accuracy. Overall, this method appears to be effective when paired with a KNN model, but as more features get added, the less feasible it becomes due to its computation time.

3.2.4 Test 4: Convolutional Neural Network (CNN):

The final test we did was using a CNN. The model that produced the best accuracy results started with an *imageInputLayer*. That layer was followed by a set of 4 layers that was repeated 3 times. A *convolution2dLayer*, a *batchNormalizationLayer*, which normalized the layers and sped up the training process, an activation function layer, and a *maxPooling2dLayer*. Finally, there was a *fullyConnectedLayer* with an output size of 2, one for each class, a *softmaxLayer*, and a *ClassificationLayer*. ("Create Simple Deep Learning Neural Network for Classification", n.d.).

After tinkering with the exact hyperparameter values, an optimal training method was created. The model was trained using stochastic gradient descent, with 7 epochs (training cycles on the whole training set). The data was shuffled at the start of every epoch and the network was re-validated every 30 iterations (**"**Create Simple Deep Learning Neural Network for Classification", n.d.).

Model	Training Time	Accuracy
CNN	347.5 sec	98.7%

Overall, the CNN performed the best out of all the tests. It scored almost a full percent higher than the next best result. The main downside of this method is that it takes a lot of time to train. One remedy to that issue, however, is that the model can be reused since new brain scans would be similar to the training data. Reusing the model means that it does not need to be retrained every time new data is received. It can be trained once and then reused on new data. This is incredibly beneficial, as unlike the previous two tests, the time-consuming part was the training, as opposed to the pre-processing. It also performs a few points better than the first test.

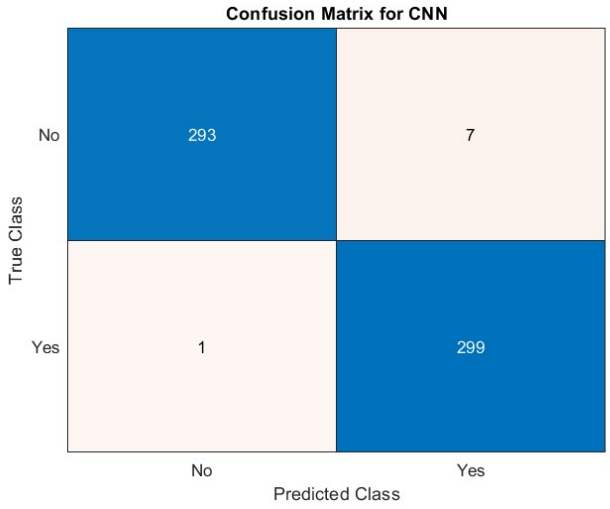

Figure 19: A confusion matrix for the CNN.

From the above, we see that there is only one false negative, an image that was classified as not having a tumour but had a tumour. This is a promising result, as when it comes to cancer detection, false positives are preferred to false negatives.

4. Conclusion

4.1 Summary of Results

Overall, machine learning seemed to be very well-equipped for detecting brain tumours from the images. Without any extra pre-processing, both the support vector machine and k-nearest neighbour models scored over 96% on the data. This created a baseline that we used to compare the other values to, since this method would be the fastest to train and test as the data only needed to be read in and reshaped, something that was already necessary to run the two models.

Then we used principal component analysis on the data, using it to reduce the dimensionality of the data. At it's very best, PCA scored slightly lower than the first test with the SVM but scored slightly higher with the KNN model. The SVM, however, did perform well when we reduced the number of components, whereas the KNN had much more consistent results at the different levels.

The final type of pre-processing we tested was non-negative matrix factorization. This method was very inefficient, as the factorization computation time exponentially increased as we added features. The SVM model performed very poorly with this method, whereas the KNN performed very well, doing better than in the two previous tests. The problem with this method, however, is that unless if you create factorized matrices with a small number of features, the process takes a very long time and does not justify the slight increase in accuracy compared to the other methods.

The final model created was a convolutional neural network, which outperformed the rest of the models when it came to accuracy, scoring 98.7%. To put that result into perspective, it means that out of the 600 test images, 592 of them were correctly classified. Thankfully, out of the 8 misclassified images, only one was a false negative. The main downside to this model was the long training time, however, since the data it has been trained on should resemble any new data coming in, this model should be able to be re-used without training again and still provide accurate results.

Overall, this report received some promising results, however, there is still room for improvement. Since many thousands of people are diagnosed with brain tumours each year and many thousands more are tested, even if a small number of scans are inaccurately classified, especially if they are any false negatives, it can be very problematic. Therefore, more work should be done in the future to improve on our results.

4.2 Comparisons to Previous Work

There are several people who have worked with this dataset and posted their results to Kaggle. For example, user Mahmoud Ali (2023) created a CNN for the data and got a test accuracy of 98.8%, practically identical to the best result we had. Other users, such as Mohsen & Elsayed (2023) managed to train a CNN that achieved well over 99% accuracy. There were many such examples of high accuracy, however, one, by user Gerry (2021) used an EfficientNetB4 model in Python to get 99.7% accuracy, with just one out of 300 test images being incorrectly classified. Beyond these results, other users also scored lower in terms of accuracy than we did. Overall, we are satisfied with our results compared to others, as our results were practically on par with the best outcomes.

4.3 Problems Encountered

There were some issues that we encountered. The first one, as mentioned before, was that the images were all different sizes, so we could not use them initially. Unfortunately, when cropping them, we had to choose between either some images being distorted, or most being cut off. We decided to go with the distorted option, which may have caused some problems as a result (see Figure 18). Additionally, some images had to be shrunken down substantially, meaning that detail was lost, which may have impacted our results. The other major issue was that loading in the images and assigning labels to them was very complicated. For the SVM and KNN methods to work, there must be a numerical target variable present. The images did not have that, so we had to manually assign classifiers. This was done by sequentially adding the images by switching between positive and negative scans, and at the same time, manually assigning labels so that the target vector was: [1; 0; 1; 0; …]. This approach meant that we had to be very careful to ensure that nothing got out of order at any point as that would have had detrimental impacts on our results. Since we got results the 90%’s for accuracy in each test, it appears that we succeeded in keeping the order.

4.4 Future Work

In the future, some ways to try and improve on our results would be to create an EfficientNet model for the data, to attempt to emulate some of the best results from the previous works. Additionally, in working with supervised learning methods, some other models could be tried such as a binary classification tree, naïve Bayes, K-means clustering, or logistic regression. These models could potentially improve upon the results seen from the SVM and KNN models.

4.5 Final Remarks

Overall, this project gave many positive results and showed a lot of promise in how machine learning can be used to detect cancer from brain scans. Despite some troubles early on, we were able to get fairly accurate results at every stage and have demonstrated that even low-level machine learning algorithms can lead to impressive results with minimal pre-processing. One of the biggest takeaways from this is the success of the first test, whereby simply feeding the reshaped data, we were able to produce very accurate diagnoses. This further proves that machine learning is a very powerful tool for cancer detection, especially in the brain.

References

11.1 - principal component analysis (PCA) procedure: Stat 505. Penn State: Statistics Online Courses. (n.d.). Retrieved April 25, 2023, from https://online.stat.psu.edu/stat505/lesson/11/11.1

Ali, M. (2023, February 19). *Brain_tumor_detection_simple CNN*. Kaggle.com. https://www.kaggle.com/code/mahmouds312s/brain-tumor-detection-simple-cnn

Bennett, K. P. & (2000, December). *Support Vector Machines: Hype or Hallelujah*. Retrieved April 26, 2023, from https://www.kdd.org/exploration_files/bennett.pdf.

Brain, other CNS and intracranial tumours statistics. (n.d.). Cancer Research UK. Retrieved April 26, 2023, from https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/brain-other-cns-and-intracranial-tumours#heading-Four

Brownlee, J. (2019, April 22). *A gentle introduction to pooling layers for Convolutional Neural Networks*. MachineLearningMastery.com. Retrieved April 25, 2023, from https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

Brownlee, J. (2020, August 14). *Support Vector Machines for machine learning*. MachineLearningMastery.com. Retrieved April 26, 2023, from https://machinelearningmastery.com/support-vector-machines-for-machine-learning/

Brunton, S. (2021, January 31). *SVD: Image Compression [Matlab]*. Www.youtube.com. https://www.youtube.com/watch?v=QQ8vxj-9OfQ&ab_channel=SteveBrunton

Cancer Research UK. (n.d.). *Cancer survival statistics*. Cancer Research UK. https://www.cancerresearchuk.org/health-professional/cancer-statistics/survival#heading-One

Compute confusion matrix for classification problem - MATLAB confusionmat. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/stats/confusionmat.html

Convert numbers to character array - MATLAB num2str. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/matlab/ref/num2str.html

Create Simple Deep Learning Network for Classification - MATLAB & Simulink Example. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html

Datastore for image data - MATLAB. (n.d.). Www.mathworks.com. Retrieved April 26, 2023, from https://www.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html

Display Grayscale, RGB, Indexed, or Binary Image. (2016). Mathworks.com. https://www.mathworks.com/help/images/ref/imshow.html

Display Multiple Images - MATLAB & Simulink. (n.d.). Www.mathworks.com. Retrieved April 26, 2023, from https://www.mathworks.com/help/images/display-multiple-images.html

Dubey, A. (2022, September 18). *The mathematics behind Principal Component Analysis*. Medium. Retrieved April 25, 2023, from https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643.

Fit k-nearest neighbor classifier - MATLAB fitcknn. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/stats/fitcknn.html

Non-Negative Matrix Factorization (2023, March 2). GeeksforGeeks. Retrieved April 25, 2023, from https://www.geeksforgeeks.org/non-negative-matrix-factorization/.

Gandhi, R. (2018, June 7). *Support Vector Machine — Introduction to Machine Learning Algorithms*. Towards Data Science; Towards Data Science.

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Gerry. (2021, November 16). *EfficientNetB4 Acc score=99.67%*. Kaggle.com. https://www.kaggle.com/code/gpiosenka/efficientnetb4-acc-score-99-67

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Deep Learning. Retrieved April 26, 2023, from https://www.deeplearningbook.org/contents/convnets.html.

Gray colormap array - MATLAB gray. (n.d.). Www.mathworks.com. Retrieved April 26, 2023, from https://www.mathworks.com/help/matlab/ref/gray.html

Guo, Gongde & Wang, Hui & Bell, David & Bi, Yaxin. (2004). *KNN Model-Based Approach in Classification.*

Hamada, A. (n.d.). *Br35H:: Brain Tumor Detection 2020*. Www.kaggle.com. https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection

Hui, J. (2023, January 10). *Machine learning - singular value decomposition (SVD) & principal component analysis (PCA)*. Medium. Retrieved April 26, 2023, from https://jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca-1d45e885e491

Jaadi, Z. (2023, March 29). *A step-by-step explanation of principal component analysis (PCA)*. Built In. Retrieved April 25, 2023, from https://builtin.com/data-science/step-step-explanation-principal-component-analysis

Johns Hopkins Medicine. (n.d.). *Brain Tumor*. Www.hopkinsmedicine.org. https://www.hopkinsmedicine.org/health/conditions-and-diseases/brain-tumor

Karimi, A. (2022, November 9). *Using a hard margin vs. soft margin in SVM*. Baeldung on Computer Science. Retrieved April 26, 2023, from https://www.baeldung.com/cs/svm-hard-margin-vs-soft-margin

Khandelwal, D. (2020, May 24). *Covariance, correlation, R squared*. Medium. Retrieved April 25, 2023, from https://medium.com/swlh/covariance-correlation-r-sqaured-5cbefc5cbe1c.

Kmhkmh. (2018, March 23). *File: euclidean distance 2D.SVG*. Wikimedia Commons. Retrieved April 25, 2023, from https://commons.wikimedia.org/wiki/File:Euclidean_distance_2d.svg.

Lee, D. D., & Seung, H. S. (n.d.). *Algorithms for non-negative matrix factorization - proceedings.neurips.cc*. Retrieved April 26, 2023, from https://proceedings.neurips.cc/paper_files/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf.

Lui, S. H. (2023). *Optimization Lecture Notes*. University of Manitoba.

MLMath.io. (2019, February 16). *Math behind support Vector Machine (SVM)*. Medium. Retrieved April 25, 2023, from https://ankitnitjsr13.medium.com/math-behind-support-vector-machine-svm-5e7376d0ee4d.

Mohsen, R., & Elsayed, T. (2023, January 6). *Brain_Tumor_Detection*. Kaggle.com. https://www.kaggle.com/code/reemtarekmohsen/brain-tumor-detection

Motulsky, H. J. (n.d.). *Graphpad prism 9 statistics guide - selection of components*. Prism 8 User Guide. Retrieved April 25, 2023, from https://www.graphpad.com/guides/prism/latest/statistics/stat_pca_process_selection_of_components.htm

Muhamad, Y., S., I., & Casi., S. (2019, May). *Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail*. ResearchGate. Retrieved April 26, 2023, from https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

Nonnegative matrix factorization - MATLAB nnmf. (n.d.). Www.mathworks.com. Retrieved April 26, 2023, from https://www.mathworks.com/help/stats/nnmf.html

O'Shea, K., & Nash, R. (2015, December 2). *An introduction to Convolutional Neural Networks*. arXiv.org. Retrieved April 26, 2023, from https://arxiv.org/abs/1511.08458

Platt, J. C. (1998, July). *Sequential minimal optimization: A fast algorithm for training support ...* ResearchGate. Retrieved April 27, 2023, from https://www.researchgate.net/publication/2624239_Sequential_Minimal_Optimization_A_Fast_Algorithm_for_Training_Support_Vector_Machines

Principal component analysis of raw data - MATLAB pca. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/stats/pca.html

Radhika. (2020, December 28). *The mathematics behind support Vector Machine Algorithm (SVM)*. Analytics Vidhya. Retrieved April 25, 2023, from https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/

Random permutation of integers - MATLAB randperm. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/matlab/ref/randperm.html

Raschka, S. (2018). STAT 479: Machine Learning Lecture Notes. Retrieved April 27, 2023, from https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/02_knn_notes.pdf

Rishabh Misra. (2019, April 30). *Support Vector Machines—Soft Margin Formulation and Kernel Trick*. Medium; Towards Data Science. https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

Ritchie, H., Roser, M., & Spooner, F. (2019, December). *Causes of Death*. Our World in Data. https://ourworldindata.org/causes-of-death#what-do-people-die-from

Singh, V. (2022, December 5). *All about Manhattan Distance*. Discover Colleges, Courses & exams for Higher Education in India. Retrieved April 25, 2023, from https://www.shiksha.com/online-courses/articles/all-about-manhattan-distance/

Singular value decomposition - MATLAB svd. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/matlab/ref/double.svd.html

Skalski, P. (2019, April 12). *Gentle dive into math behind Convolutional Neural Networks.* Medium. Retrieved April 25, 2023, from https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9

Surampudi, S. (2017, May 15). *Data Mining Concepts. Non-Negative Matrix Factorization.* Retrieved April 25, 2023, from https://docs.oracle.com/database/121/DMCON/GUID-76F89641-E1D3-4B11-8319-4A152389D510.htm#DMCON566

Tests for Brain and Spinal Cord Tumors in Adults. (2020, May 5). Cancer.org; American Cancer Society. https://www.cancer.org/cancer/brain-spinal-cord-tumors-adults/detection-diagnosis-staging/how-diagnosed.html

Train support vector machine (SVM) classifier for one-class and binary classification - MATLAB fitcsvm. (n.d.). Www.mathworks.com. https://www.mathworks.com/help/stats/fitcsvm.html

Tumour. (n.d.). Oxfordlearnersdictionaries.com. Retrieved April 26, 2023, from https://www.oxfordlearnersdictionaries.com/definition/english/tumour

Unzueta, D. (2022, October 18). *Fully connected layer vs. convolutional layer: Explained*. Built In. Retrieved April 26, 2023, from https://builtin.com/machine-learning/fully-connected-layer

Vinh-Trung, L., Germain, F., Jonathan, W., Paul, B., Fahima, D., & Pierre-Alain, M. (2020). *Example of euclidean and Manhattan distances between two points A and ...* Retrieved April 26, 2023, from https://www.researchgate.net/figure/Example-of-Euclidean-and-Manhattan-distances-between-two-points-A-and-B-The-Euclidean_fig8_333430988

What is the K-nearest neighbors algorithm? IBM. (n.d.). Retrieved April 25, 2023, from https://www.ibm.com/topics/knn

Wikimedia Foundation. (2022, January 27). *Sequential minimal optimization*. Wikipedia. Retrieved April 26, 2023, from https://en.wikipedia.org/wiki/Sequential_minimal_optimization

Wu, J. (2017, May 1). *Introduction to convolutional neural networks - NJU*. Introduction to Convolutional Neural Networks. Retrieved April 26, 2023, from https://cs.nju.edu.cn/wujx/paper/CNN.pdf

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018, June 22). *Convolutional Neural Networks: An overview and application in radiology - insights into imaging*. SpringerLink. Retrieved April 25, 2023, from https://link.springer.com/article/10.1007/s13244-018-0639-9

Zhang, Z. (2016, June). *Introduction to machine learning: K-Nearest Neighbors*. Annals of translational medicine. Retrieved April 25, 2023, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4916348/