# APPLIED MACHINE LEARNING SYSTEM ELEC0134 21/22 REPORT

*SN: 21207190*


## ABSTRACT[1]

Identifying (Task A) and determining the tumour types (Task B) from magnetic resonance images are binary and multiclass classification computer vision (CV) problems. The Kaggle data science platform's tumour detection dataset and a custom test set were used to train and test the machine learning (ML) models built. Linear classifiers such as support vector machines (SVMs) or logistic regressors (LR) are effective at tumour detection but cannot match state-of-the-art convolutional neural networks (CNNs) such as the residual network (ResNet) or the U-Net. The ML techniques implemented in Python with the *NumPy*, *scikit-learn*, *imbalanced-learn*, *Tensorflow* and *OpenCV* libraries were exploratory data analysis (EDA), pyramid reduction function, logistic regression, support vector classifier (SVC), principle component analysis (PCA) and CNNs. The best performing classifier for Task A and B was the CNN with 94.5% and 86.5% accuracies respectively for both tasks.


## 1. INTRODUCTION

Magnetic resonance imaging (MRI) is a safe, state-of-the-art, non-invasive biomedical imaging modality used to visualize the human body's internal tissues [1]. A tumour is an abnormal growth of cells categorized as benign, pre-malignant or malignant, which cause cancer [2]. Brain tumours are lethal, thus detecting them early increases a patient's survival rate [3]. Automatic image segmentation with deep learning (DL) became the state-of-the-art method to identify brain tumours and their type [4].

This project consisted of identifying brain tumours (Task A) and their types including meningiomas, gliomas, and pituitary tumours (Task B) shown on Figure 14 with ML models. The two CV tasks are binary (Task A) and multiclass (Task B) image classification problems. The ML models were trained on the tumour detection dataset available on the data science platform Kaggle [5] and tested on a custom test set provided for the project.

## 2. LITTERATURE SURVEY

### 2.1 Linear Classification Methods for Tumour Detection

Linear classifiers remain effective for automated MRI tumour detection and combined with data augmentation techniques such as PCA, image filters or wavelet analysis can highlight features in images [6]. ML scientists partitioned these features into three categories shape-based, intensity-based and texture-based presented on Figure 15. SVMs combined with genetic algorithms (GA) or particle swarm optimization (PSO) classify tumours with 94.4% accuracy and with PCA 96% accuracy is reachable [6].

### 2.2 Deep Learning Methods for Tumour Detection

Artificial neural networks (ANN) classify tumours with 90-100% accuracy for automated MRI brain tumour detection with PCA [6]. However, state-of-the-art CNNs are better as they extract features automatically from images without pre-processing [7]. Sophisticated CNN architectures emerged to counter the vanishing gradient problem causing weights to not update due to weight changes being too small to make a difference [8]. These architectures include the ResNet and the U-Net, which outperform most image classification but require huge computational resources for training not available in this project.

The ResNet adds a parallel feedforward path (Figure 16) or convolutional block (Figure 17) to a CNN's convolutional block connecting the input to the last flattening layer of the CNN's main path, which stops the gradient from vanishing [9]. The ResNet-50's (Figure 18) accuracy for tumour identification is 90% [9].

The U-Net employs the skip architecture in Figure 19, which combines the high-level representation of deep decoding layers with the appearance representation of shallow encoding layers for image segmentation. This architecture consists of a downsampling path decreasing the feature maps' sizes and an upsampling path increasing the feature maps' sizes. The U-Net's accuracy for MRI tumour detection is 86% [10].

## 3. DESCRIPTION OF MODELS

### 3.1 Exploratory Data Analysis

Before applying ML techniques, the dataset should be explored [11] to ensure class imbalance is prevented [12]. This imbalance was rectified using the synthetic minority oversampling technique (SMOTE) to oversample the "No Tumour[2]" minority class [13] as under sampling the other

[1] The code used to generate the experiments is available on GitHub at the following link: https://github.com/neurogeek953/AMLS_21-22_SN21207190.

[2] Note in the code the "no tumour" class is referred to as "no tumor" like in the code with American spelling in the implementation section.

classes could suppress essential data required for training the classifier [12].

### 3.2 Pyramid Reduction Function

Reducing the image size with the pyramid reduction function is essential to ensure training time is not too extensive [14]. An image pyramid is a data structure representing an image and accelerating feature extraction algorithms [15]. The pyramid reduction function REDUCE is [16]:

$$g_l(i,j) = \sum_{n=-M_l}^{M_l} \sum_{n=-M_l}^{M_l} w_l(m,n) g_{l-1}(2i+m, 2j+n) \quad (3.1)$$

### 3.3 Task A: Binary Classification

#### *3.3.1 Logistic Regression*

A LR is a probabilistic classifier computing the conditional probability $P(y|\underline{x})$ an event $y$ happens if an event $\underline{x}$ happened with the sigmoid function (Equation 3.2). $\underline{x}$ is the vector of event features and $\underline{w}$ is the model's weights and $n$ the number of features [17].

$$P(y|\underline{x}) = \frac{1}{1 + \exp(-\underline{w}^T \underline{x})} \quad (3.2)$$

The LR is trained by optimizing the negative log-likelihood (NLL) or the cross-entropy (Equation 3.3) error with a gradient descent method, where $N$ is the number of samples, $y$ the label and $\mu$ the predicted label [17].

$$NLL(\underline{w}) = -\sum_{i=1}^{N} [y_i log(\mu_i) + (1 - y_i) log(1 - \mu_i)] \quad (3.3)$$

The weight change and weights are calculated with Equations 3.4 and 3.5 [17], where $\alpha$ is the learning rate and $i$ the iteration number:

$$\Delta w = \frac{df(\underline{w})}{d\underline{w}} = \sum_i (y_i - \mu_i) \underline{x}_i \quad (3.4)$$

$$w_{i+1} = w_i + \alpha \Delta w \quad (3.5)$$

#### *3.3.2 Support Vector Machines*

SVMs can be combined with optimization methods such as GAs or PSOs and data augmentation methods including PCA [18]. The SVC's objective is to maximize the distances between classes by maximizing the decision margin $\rho$ between the two classes and the convex function $Q(\alpha)$ [19]:

$$\rho = \frac{2}{\|\underline{w}\|} \quad (3.6)$$

$$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \underline{x}_i^T \underline{x}_j \quad (3.7)$$

Two boundary conditions are applied to $Q(\alpha)$, where $y_i$ is the label of the $i^{th}$ sample, $\underline{x}_i$ its features:

1. $\sum \alpha_i y_i = 0$
2. $\alpha_i \geq 0, \forall \alpha_i$

This optimization is solved with the Lagrange multiplier to find the weights $\underline{w}$ and the bias b of the hyperplanes or lines that separate the classes best.

The SVC's hyperparameters are tuned with grid search (GS) to find the best parameter combination inside a parameter space by minimizing its expected loss [20].

#### *3.3.3 Principle Component Analysis (PCA)*

PCA is a dimensionality reduction algorithm used to extract the directions or principle components along which the data varies the most [21]. The PCA algorithm optimizes Equation 3.7, where $U$ is the recovery, $W$ the projection matrix, $\underline{x}_i$ a data sample and $m$ the number of samples:

$$argmin \left( \sum_{i=1}^{m} \|\underline{x}_i - UW\underline{x}_i\| \right) \quad (3.8)$$

The algorithm has five steps: The data's average is calculated, then the covariance matrix is computed. The eigenvalues and eigenvectors, which are the principle components are computed. Finally, the principle components with the largest eigenvalues are selected to represent the data.

### 3.4 Task B: Multiclass Classification

#### *3.4.1 Convolutional Neural Networks (CNN)*

A CNN is an ANN designed to recognize visual patterns from image pixels [22]. CNNs have at least 1 convolutional, 1 pooling, 1 fully-connected, 1 hidden and 1 output layer [23]. These layers extract "learnable kernels" from images stacked into a two-dimensional (2D) activation map [23]. Then, these kernels are assigned to nodes in the convolutional layer conducting rectified correlations on a sphere (RECOS) to compute the similarity image fragments share with the features in the activation map. The latter are weighted during training via backpropagation [22] and their dimensions are reduced in the pooling layer connected to the ANN producing the final result [23]. In practice, convolutional, pooling and

fully connected layers are stacked to improve results as shown in Figure 20 [24].

CNNs use the sigmoid (Equation 3.9), the SoftMax (Equation 3.10) and the rectifier linear unit (ReLu) (Equation 3.11) activation functions [25]:

$$f(x) = \frac{1}{1 - e^{-x}} \tag{3.9}$$

$$f(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \, , for \; i = 1, \dots, K \tag{3.10}$$[3]

$$f(x) = \max(0, x) \tag{3.11}$$

*3.4.2 Dropout*

Dropout is a regularization method for ANNs to fight overfitting by breaking the co-adaptation of feature detectors as dropped-out neurons cannot influence the retained neurons. Dropout assigns a dropout probability to each layer and then randomly sets the weights of chosen nodes to 0 [26].

## 4. IMPLEMENTATION[4]

### 4.1 Data Pre-processing: Function *preprocess_data*

Data pre-processing is handled by the *preprocess_data* function located in the files *trainCNN.py* and *trainLinearClassifiers.py*, which takes as arguments the file path to the training or test set and the string *"smote"* to select the SMOTE oversampling method. A second dataset is created by binarizing the original labels into the *"tumor"* and *"no_tumor"* categories and the image file names are extracted. The latter are label encoded using the *LabelEncoder* class from the *pre-processing* module in the *scikit-learn* library. Then the *SMOTE* class in the *imbalanced-learn* library's *oversample* module creates a new dataset with the original labels and another with binarized labels, where the *"no_tumor"* minority class is oversampled to contain the same number of samples as the majority class. The original image file names are recovered in both datasets and the labels for all datasets are one hot encoded [27] with *preprocessing*'s *OneHotEncoder* class. Finally, the function returns the encoders and each dataset's inputs and outputs.

### 4.2 EDA: File *ExploratoryDataAnalysis.py*

EDA is conducted in the *ExploratoryDataAnalysis.py* file's by calling the *preprocess_data* function imported from the *TrainCNN.py* file on the training and test sets. Both datasets are explored with the *exploratory_data_analysis* and *explore_test_set* functions. Both functions call the *count_binary_data* and the *count_data* functions to count the elements of each class for both datasets and the resampled ones. Finally, the function *plot_histograms* and *plot_original_histogram* function use *matplotlib* library's *pyplot* module to plot the data distribution for the resampled, binarized, original on Figure 1, and test sets on Figure 8.

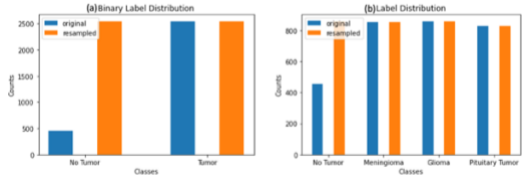


**Figure 1 Distribution of the Binary (a) and Class (b) Labels in the Original and Resampled Datasets.**

### 4.3 Task A: Binary Classification

*4.3.1 LR: Class LogisticRegressor*

The LR was implemented custom in a *LogisticRegressor* class in the file *LogisticRegression.py*. The LR is initialized as an instance with the LR's parameters as attributes of the class. These include the iteration count set to 3000, the cost sampling rate of 10, the learning rate of 0.3 and the image size of 128 by 128 pixels or 2048 principle components corresponding to 16384 or 2048 weights. The LR is trained in the *trainLogisticRegressor* method, which calls the *optimize* and *sigmoid* methods and uses *pyplot* to plot the NLL on Figure 2. The *evaluateLogisticRegressor* method calling the predict method rounding up the output if its greater than 0.5 and gets the accuracy metrics of the LR.

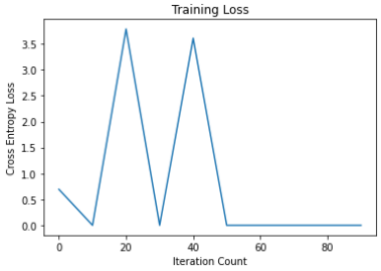


**Figure 2 LR Training Loss: First 100 Iterations.**

*4.3.2 SVC & GS: Class SupportVectorClassifier*

The *SupportVectorClassifier* class wraps the *scikit-learn* the *GridSearch* and *SVC* classes together in the *SVMClassifier.py* file. Thus, the SVC and GS's parameter space contains the potential the $C$-coefficients of 0.1, 1, 10, 100, $\gamma$-coefficients 0.0001, 0.001, 0.1, 1 and the gaussian kernel. The SVC's input is a 16 by 16 image or 256 principle components to minimize the classifier's training time. The

[3] Note K is the number of classes.

[4] Note the implementations file organisation is in the Appendix on Figure 21.

SVC is trained with the *trainSupportVectorClassifier* method calling the fit method of the *GridSearch* class and computes the training predictions with the *predictWithSVC* method, which rounds up the probability to 1 if it is greater or equal to 0.5. The last method *evaluateSupportVectorClassifier* calculates the accuracy of the classifier and draws the confusion matrix. The GS chose a C of 1 and a $\gamma$ of 0.1, for the SVC and SVC-PCA and a C of 1 and a $\gamma$ of 1 for the cosine kernel SVC-PCA.

*4.2.3 PCA and Kernel-PCA: Class PrincipleComponents*

The *PrincipleComponents* class implemented in the *principleComponentAnalysis.py* file is initialized with the number of principle components and the PCA's kernel type. The methods *fitPrincipleComponentAnalysis* fits the PCA or kernel-PCA to the data and the *augmentData* method extracts features from an image. LR had the best results with a linear kernel and SVC worked best with the cosine kernel. 2048 and 256 principle components were taken for the LR and SVC.

*4.3.4 Wrapper for Linear Classifiers: Class LinearClassifiers*

The linear classifier wraps the *SupportVectorClassifier* and *LogisticRegressor* classes. It is initialized with the parameters of both classes, then trains both classifiers with their training methods inside the *LinearClassifiers*' *trainClassifiers* method. The class has an *identifyTumors* method and an *evaluateLinearClassifier* method gets the accuracy metrics of the classifiers with their evaluate methods.

*4.3.5 Training Classifiers: Function train_and_crossvalidate*

The models are trained in the *train_and_crossvalidate* function in the *TrainLinearClassifiers.py* file, which takes as arguments the path to the file names, the number of k-folds, the training set file names, the binary and multiclass labels, the SVC, LR and PCA parameters. The *preprocess_input_data* function takes as arguments the path to the file names and the file names. Then, it iterates over the data, calls the *get_image* function and turns the images into one-dimensional arrays with the *NumPy*'s *flatten* method. *get_image* uses the OpenCV package's *imread* method to get a two-dimensional grayscale image, the *resize* method to REDUCE it and normalizes the pixels between 0 and 1 by dividing by 255. Then the PCA or kernel-PCA instances augment the image if used and the *best_classifiers* instance of the *LinearClassifier* class is initialized. Then the data is split into 5 k-folds setting aside 20% of the data for the validation set by invoking twice the *KFold* class from *scikit-learn*'s *model_selection* module to train the LR and SVC models in the *LinearClassifier* instance. This instance's models are updated at each k-fold to have the best validation[5] accuracy. Then the validation accuracies over the k-folds are plotted in Figure 3 (a) and (b).

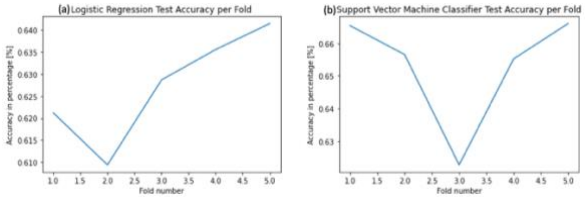


**Figure 3 LR (a) and SVC (b) Validation[6] Accuracies per K-Fold.**

*4.3.6 Classifiers Performance: Function test_and_evaluate*

The *test_and_evaluate* function in the *TrainLinearClassifiers.py* file takes as arguments the image file names in the test set, the file path to the test images, test labels, the *LinearClassifier* instance and the PCA models for the SVC and LR. Then, the *preprocess_input_data* prepares the images for the classifiers. PCA augments the data if a PCA model was passed in the function. The classifier's performances are evaluated with the *classification_report* and *confusion_matrix* functions from *scikit-learn*'s *metrics* module and by calculating the accuracy, specificity and recall. The *ConfusionMatrixDisplay* class from the *metrics* module plots the confusion matrix.

**4.4 Task A & B: DL Binary & Multiclass Classification**

*4.4.1 CNN: Class CNN_Classifier*

The CNN's implementation is in the *CNN_Classifier* class in the file CNN.py, whose attributes include the layers and input size. These attributes were shaped with the *select_1st_convolutional_layer_arguments*, *select_2nd_convolutional_layer_arguments*, *select_pooling_layer_arguments*, *select_pooling_layer_window_size_arguments* and *select_hidden_layer_arguments* functions in the *TrainCNN.py* file to initialize the layers with TensorFlow's *Keras* module. The Adam optimizer with a learning rate of 0.001 and $\beta$ -parameters of 0.9 and 0.999 applies backpropagation on the CNN. For Task A the output layer contained two output nodes and for Task B four output nodes operating with the SoftMax activation function.

**Table 1.** CNN Architecture

| Layer Number | Layer Type | Neurons | Dropout Ratio | Activation Function |
|---|---|---|---|---|
| Layer 1 | Convolutional | 4194304 | 0 | ReLu |
| Layer 2 | Pooling | 262144 | 0.25 | ReLu |
| Layer 3 | Convolutional | 524288 | 0 | ReLu |
| Layer 4 | Pooling | 131072 | 0.25 | ReLu |
| Layer 5 | Convolutional | 131072 | 0 | ReLu |

[5] The validation set here is referred to as the test set on figures.

[6] The validation accuracy is referred to as test accuracy on the figures.

| | | | | |
|---|---|---|---|---|
| Layer 6 | Pooling | 15488 | 0.3 | ReLu |
| Layer 7 | Convolutional | 15488 | 0 | ReLu |
| Layer 8 | Pooling | 3200 | 0.3 | ReLu |
| Layer 9 | Convolutional | 6400 | 0 | ReLu |
| Layer 10 | Pooling | 1024 | 0.3 | ReLu |
| Layer 11 | Flattening | 1024 | 0 | ReLu |
| Layer 12 | Hidden | 1024 | 0.5 | ReLu |
| Layer 13 | Hidden | 512 | 0.5 | ReLu |
| Layer 14 | Output | 2 or 4 | 0 | SoftMax |

The *CNN_Classifier* class has an *identify_tumor* method predicting a tumour's presence type in Task A or its type in Task B. The methods *saveCNNWeights* and *loadCNNWeights* respectively save the model as a *.h5* file and load it to ensure models are available for use and testing.

*4.4.2 CNN Training: Function train_and_crossvalidate*

The *train_and_crossvalidate* function in the TrainCNN.py file takes as arguments the file name, file path, labels, image dimensions, the batch size of 64, the number of epochs 50, the number of k-folds 5 and the layers' parameters in Table 1 for the *CNN_Classifier* instance. Image preprocessing is carried out like in the *trainLinearClassifiers.py* file's *train_and_crossvalidate* function but the image is resized to 256 by 256 by 3. 20% of the k-fold data is set aside for the validation set like in the TrainLinearClassifiers.py file and the CNN with the best validation accuracy is updated at each k-fold and its accuracy over each k-folds is plotted using *matplotlib*'s *pyplot* module in Figure 5 and Figure 7. The best CNN's training, validation accuracies and losses for Tasks A and B were plotted in Figure 4 and Figure 6. The function returns the best CNN and it is saved as a *.h5* file using with the instance's *saveCNNWeights* method in the file's main module.

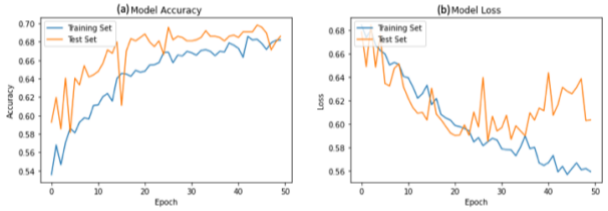


**Figure 4 CNN Training and Validation[7] Accuracies and Losses in Task A.**

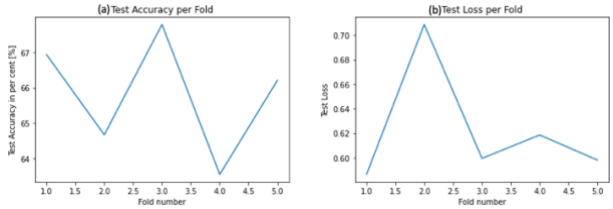


**Figure 5 CNN Validation[8] Accuracy and Loss per K-Fold in Task A.**

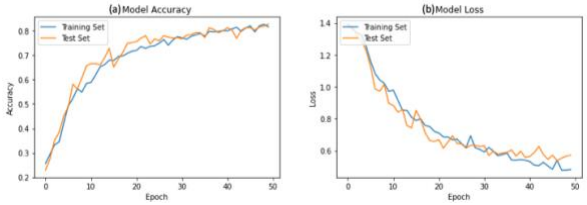


**Figure 6 CNN Training and Validation[9] Accuracies and Losses for Task B.**

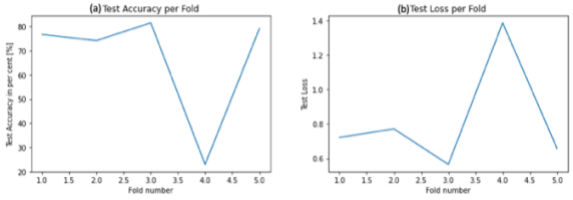


**Figure 7 CNN Validation[10] Accuracy (a) and Loss (b) per K-Fold in Task B.**

*4.4.3 CNN Performance: Function test_and_evaluate*

The *test_and_evaluate* function in the *TrainCNN.py* file computes the classifier performance for each output node type by calling the *CNN_Classifier* instance's *identifyTumors* method. Task A could be completed with 1 sigmoid output node or 2 SoftMax output nodes and the accuracy, recall and specificity and reported using the *classification_report*, *confusion_matrix* and *ConfusionMatrixDisplay* tools from scikit-learn's *metrics* module. For Task B, the CNN had 4 SoftMax output nodes and its prediction accuracy overall and for each class were computed with the tools in the *metrics* module.

*4.4.4 Test the CNN: File TestCNN.py*

The *TestCNN.py* file loads 7 trained CNNs saved in *.h5* files. The file imports the *test_and_evaluate* function from the *TrainCNN.py* file and applies it on the test set to display the models' performances.

## 5. EXPERIMENTAL RESULTS AND ANALYSIS

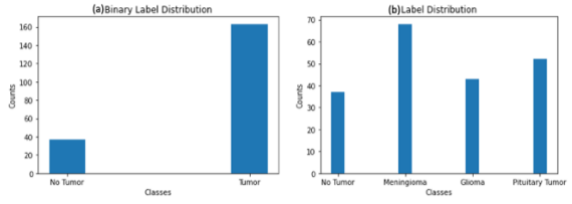


**Figure 8 Distribution of the Binary (a) and Class (b) Labels in the Test Set.**

[7] The validation set is referred to as test set on Figure 4.
[8] The validation set is referred to as test set on Figure 5.
[9] The validation set is referred to as test set on Figure 6.
[10] The validation set is referred to as test set on Figure 7.

### 5.1. Task A: Binary Classification with Linear Classifiers

**Table 2.** Linear Classifiers Test Set Performance

| Model Type | Accuracy | Recall | Specificity |
|---|---|---|---|
| LR | 87.0% | 97.5% | 40.5% |
| LR-PCA | 37.5% | 27.5% | 89.2% |
| **SVC** | **94.5%** | **94.5%** | **94.6%** |
| SVC-PCA | 89.0% | 91.4% | 78.4% |
| SVC-KPCA | 85.0% | 85.9% | 81.1% |

**Table 3.** Linear Classifiers Validation Set Performance

| Model Type | Accuracy | Recall | Specificity |
|---|---|---|---|
| LR | 64.1% | 95.6% | 33.3% |
| LR-PCA | 52.4% | 27.5% | 84.9% |
| SVC | 66.6% | 87.0% | 46.8% |
| SVC-PCA | 67.3% | 88.9% | 46.1% |
| **SVC-KPCA** | **66.6%** | **85.9%** | **50.0%** |

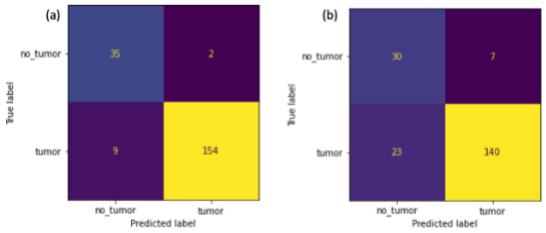


**Figure 9 SVC Confusion Matrices for the SVC (a) and Cosine Kernel PCA SVC (b) for the Test Set.**

The SVC is better than the LR classifier as it has an accuracy of 94.5% and a specificity 94.6 % on the test set versus 87% and 40.5% as shown in Table 2. When PCA[11] extracts features, the LR accuracy decreases to 37.5% and specificity increases to 89.2%. Thus, the LR is a bad classifier as it cannot have good recall and specificity at the same time. The SVC linear classifier appears to be the best but its performance on the validation set shows that it has worse a specificity than the cosine kernel-PCA SVC as shown in Table 3. This means if the test set shown on Figure 8.a was not imbalanced the SVC's performance may be lower than the SVC preceded by a cosine kernel PCA feature extractor. Thus, the cosine kernel SVC is the best linear classifier for Task A.

### 5.2 Task A: Binary Classification with CNN

**Table 4.** CNN Binary Classification Test Set Performance

| Output Neurons | Activation Function | Epochs | Dropout | Accuracy | Recall | Specificity |
|---|---|---|---|---|---|---|
| 1 | Sigmoid | 50 | No | 18.5% | 0.0% | 100.0% |
| 1 | Sigmoid | 50 | Yes | 18.5% | 0.0% | 100.0% |
| 2 | SoftMax | 50 | No | 65.5% | 57.7% | 100.0% |
| 2 | SoftMax | 50 | Yes | 92.0% | 92.6% | 89.2% |
| **2** | **SoftMax** | **100** | **Yes** | **94.5%** | **97.3%** | **93.9%** |

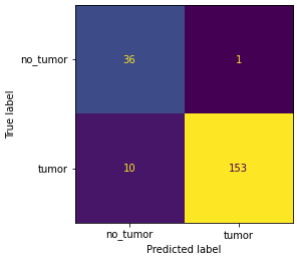


**Figure 10 Best CNN's Test Set Confusion Matrix.**

The most important hyperparameter is the number of output nodes as if the CNN has one sigmoid output neuron it will not train as shown on Figure 11 even with dropout. Changing the output nodes to two SoftMax neurons causes the accuracy to jump from 18.5% to 65.5% as observed on Table 4. This performance is pushed to a 92% accuracy with dropout, which prevents overtraining as the validation accuracy increases with the training accuracy on Figure 12.b as opposed to Figure 12.a. Doubling the number of epochs slightly increases accuracy to 94.5% and adding or subtracting hidden layers did not cause significant changes in model performance. Thus, DL methods are more reliable than their linear ML counterparts as their specificity and recall remain consistent.

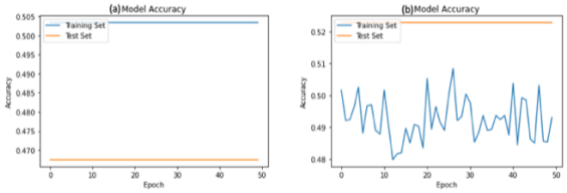


**Figure 11 CNN 1 Sigmoid Output Neuron[12] Accuracy Trained without Dropout (a) and with Dropout (b).**

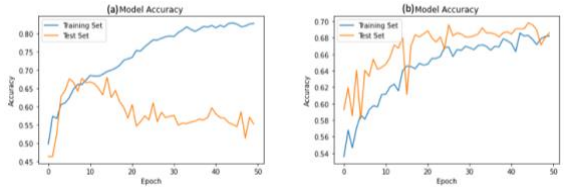


**Figure 12 CNN 2 SoftMax Output Neurons[13] Accuracy Trained without Dropout (a) and with Dropout (b).**

### 5.3 Task B: Multiclass Classification with CNN

**Table 5.** CNN Classification Test Set Accuracies

| Dropout | Overall | No Tumor | Meningioma | Glioma | Pituitary Tumor |
|---|---|---|---|---|---|
| **No** | **86.5%** | **74.0%** | **90.0%** | **90.0%** | **91.0%** |
| Yes | 85.0% | 80.0% | 89.0% | 86.0% | 84.0% |

[11] The linear PCA kernel was the best for the LR, thus other results were not shown due to being worse.

[12] The validation set is referred to as test set on Figure 11.

[13] The validation set is referred to as test set on Figure 12.

In Task B, the CNN trained without dropout is the best as it differentiates better between the different types of tumours as shown on Table 5, which is probably due to overfitting as Figure 13.a shows a gap of 10% between training and validation accuracy. When trained with dropout, this gap vanishes as seen Figure 6.a, which explains this CNN's aptitude to better recognize healthy patients.

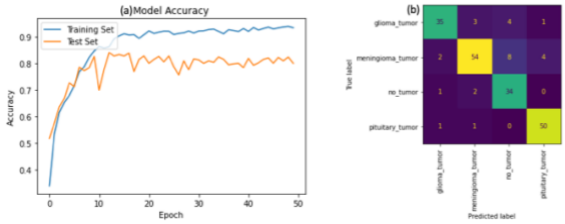


**Figure 13 CNN 4 SoftMax Output Neurons Trained Accuracy without Dropout Accuracy (a) [14] and the Resulting Confusion Matrix (b).**

### 5.4 Discussion

The test set's data class distribution is similar to the original dataset's class distribution even when binarized, which may create unnaturally good results. To check if the models learnt the tumour concept, noise should be introduced in the test set to see if the results remain consistent. Previous experiments have shown applying noise significantly reduced the ability of a classifier to identify tumours and their type [28]. In this case, the result would be a big accuracy drop for linear classifiers and a less pronounced one for the CNN as it uses "learnable kernels" as opposed to statistical learning.

Image filters such as wavelets, Gabor filters and edge enhancers can highlight regions of interests in images. Research suggests it is easier for classifiers to learn features and recognize them in the test set when highlighted [29]. This would result in higher test and training accuracies.

## 6. CONCLUSION

CNNs remain the best model type to identify tumours and their type. The model accuracy in task A was 94.5% with excellent recall and specificity. In Task B, the multiclass counterpart achieves an accuracy of 86.5 % without dropout and identifies the tumour type with a 90.3% accuracy. Linear classifiers in particular the SVC combined with GS remained effective on the test set with an accuracy of 94.5% but its validation accuracy of 66.6% is clearly inferior.

Further work, should explore the performance of the U-Net or ResNet-50 using high performance computing (HPC) services such as AWS, Azure or UCL's HPC. The latter CNNs cannot be tested on a laptop with 16 gigabytes RAM and no Nvidia GPUs, which means CUDA accelerators are not accessible. Thus, the project's scope was limited.

[14] The validation set is referred to as test set on Figure 13.a

## 7. REFERENCES


[1] O. N. Jaspan, R. Fleysher and M. L. Lipton, "Compressed sensing MRI: a review of the clinical literature" *British Journal of Radiology*, vol. 88, 2015.

[2] C. Chun, "What are the different types of tumor?" 21 August 2019. [Online]. Available: https://www.medicalnewstoday.com/articles/249141.

[3] A. Isin, C. Direkoglu and M. Sah, "Review of MRI-based brain tumor image segmentation using deep learning methods" in *12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016, 29-30 August 2016,*, Vienna, Austria, 2016.

[4] M. Nazir, S. Shakil and K. Khurshid, "Role of deep learning in brain tumor detection and classification (2015 to 2020): A review" *Computerized Medical Imaging and Graphics*, vol. 91, pp. 1-30, 2021.

[5] S. Bhuvaji, A. Kadam, P. Bhumkar, S. Dedge and S. Kanchan, "Brain Tumor Classification (MRI)." 2020. [Online]. Available: doi: 10.34740/KAGGLE/DSV/1183165.

[6] M. A. Reema, A. Prasad and A. P. Babu, "A Review on Feature Extraction Techniques for Tumor Detection and Classification from Brain MRI" in *International Conference on Intelligent Computing,Instrumentation and Control Technologies (ICICICT)*, Kerala, 2017.

[7] Floreano Dario and Mattiussi Claudio, Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies, 1st Edition ed., London: MIT Press, 2008.

[8] Z. Hu, J. Zhang and Y. Ge, "Handling Vanishing Gradient Problem Using Artificial Derivative" *IEEE*, vol. 9, pp. 22371-22377, 2021.

[9] L. H. Shehab, O. M. Fahmy, S. M. Gasser and M. S. El-Mahallawy, "An efficient brain tumor image segmentation based on deep residual networks (ResNets)" *Journal of King Saud University - Engineering Sciences*, vol. 33, no. 6, pp. 404-412, 2021.

[10] H. Dong, G. Yang, F. Liu, . Y. Mo and Y. Guo, "Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks" 3 June 2017. [Online]. Available: https://arxiv.org/pdf/1705.03820.pdf. [Accessed 28 November 2021].

[11] A. T. Jebb, S. Parrigon and S. E. Woo, "Exploratory data analysis as a foundation of inductive research"

*Human Resource Management Review*, vol. 27, pp. 265-276, 2017.

[12] M. Roweida, J. Rawashdeh and M. Abdullah, "Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results" in *2020 11th International Conference on Information and Communication Systems (ICICS)*, Irbid, 2020.

[13] B. Santoso, H. Wijayanto and B. Sartono, "Synthetic Over Sampling Methods for Handling Class Imbalanced Problems : A Review" in *IOP Conference Series: Earth and Environmental Science*, Bogor, Indonesia, 2017.

[14] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, *Pyramid methods in image processing*, Princeton: RCA Laboratories, 1984.

[15] R. Szeliski, Computer Vision: Algorithms and Applications, 1st Edition ed., Washington: Springer, 2011.

[16] P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a Compact Image Code" *IEEE Trans. Communications*, vol. 9, no. 4, pp. 532-540, 1983.

[17] K. P. Murphy, Machine Learning : A Probabilistic Perspective, Cambridge: MIT Press, 2012.

[18] J. Nalepa and M. Kawulok, "Selecting training sets for support vector machines: a review" *Artificial Intelligence Review*, vol. 52, p. 857–900, 2019.

[19] N. Deng, . Y. Tian and C. Zhang, Support Vector MachinesOptimization Based Theory, Algorithms, and Extensions, Boca Raton: CRC Press, 2012.

[20] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization" *Journal of Machine Learning Research*, vol. 13, pp. 281-305, 2012.

[21] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning, New-York: Cambridge University Press, 2014.

[22] C.-C. J. Kuo, "Understanding Convolutional Neural Networks with A Mathematical Model" *Journal of Visual Communication and Image Representation*, vol. 41, no. 1, pp. 406-413, November 2016.

[23] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks" Cornell University Library, New-York, 2015.

[24] S. Albelwi and A. Mahmood, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks" *Entropy*, vol. 19, no. 6, pp. 1-10, 24 May 2017.

[25] S. Sharma, . S. Sharma and A. Athaiya, "ACTIVATION FUNCTIONS IN NEURAL NETWORKS" *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, pp. 310-316, 2020.

[26] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks" *Neural Networks*, vol. 71, pp. 1-10, 2015.

[27] J. Brownlee, "Why One-Hot Encode Data in Machine Learning?" 30 June 2020. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. [Accessed 1 December 2021].

[28] S.-C. Wen, . Y.-J. Chen, Z. Liu, W. Wen, X. Xu, Y. Shi, T.-Y. Ho, Q. Jia, M. Huang and J. Zhuang, "Do Noises Bother Human and Neural Networks In the Same Way? A Medical Image Analysis Perspective" 4 November 2020. [Online]. Available: https://arxiv.org/pdf/2011.02155.pdf. [Accessed 12 December 2021].

[29] M. Kociołek, M. Strzelecki and R. Obuchowicz, "Does image normalization and intensity resolution impact texture classification?" *Computerized Medical Imaging and Graphics*, vol. 81, pp. 1-17, 2020.

[30] J. Cheng, W. Huang, S. Cao, R. Yang, W. Yang, Z. Yun, Z. Wang and Q. Feng, "Enhanced Performance of Brain Tumor Classification via Tumor Region Augmentation and Partition" *PLOS ONE*, vol. 10, no. 12, pp. 1-13, 2015.

[31] K. Gurney, An Introduction to Neural Networks, CRC Press Taylor&Francis Group, 1997.

[32] S. Gasparini and M. Migliore, "Action Potential Backpropagation" in *Encyclopaedia of Computational Neuroscience*, D. Jaeger and R. Jung, Eds., New York, New York: Springer, 2014.

[33] J. Patterson and A. Gibson, Deep Learning: A practitioners approach, Sebastopol, California: O'Reilly Media Inc., 2017.

[34] L. Tarassenko, A Guide to Neural Computing Applications, 1st Edition ed., London: Elsevier Inc., 1998.

## 8. APPENDIX

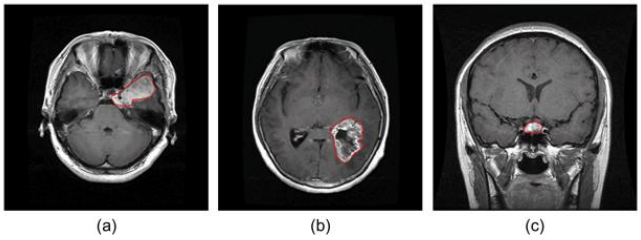


**Figure 14 Different Types of Brain Tumours in the Dataset: (a) Meningioma (b) Glioma (c) Pituitary Tumor [30].**

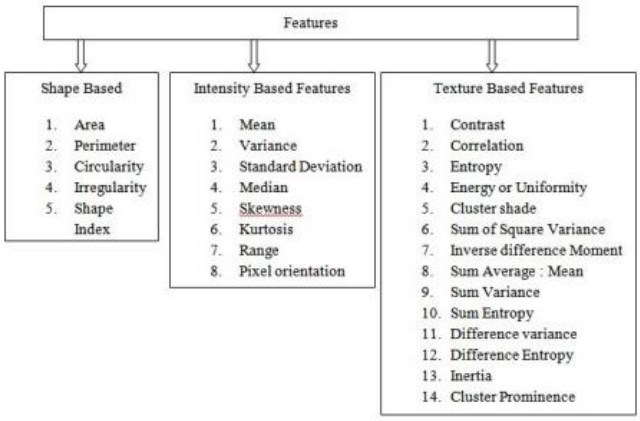


**Figure 15 Types of Features in Images [6].**

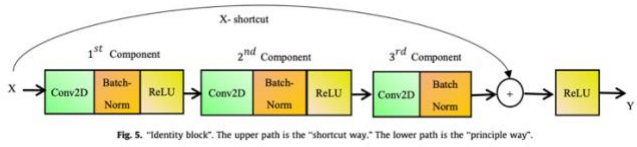


**Figure 16 ResNet Identity Block [9].**

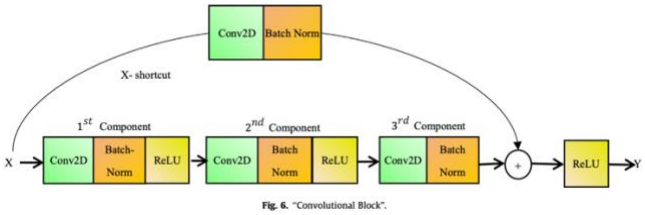


**Figure 17 ResNet Convolutional Block [9].**

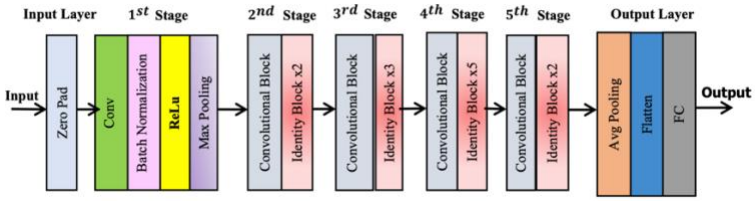


**Figure 18 ResNet50 Architecture [9].**

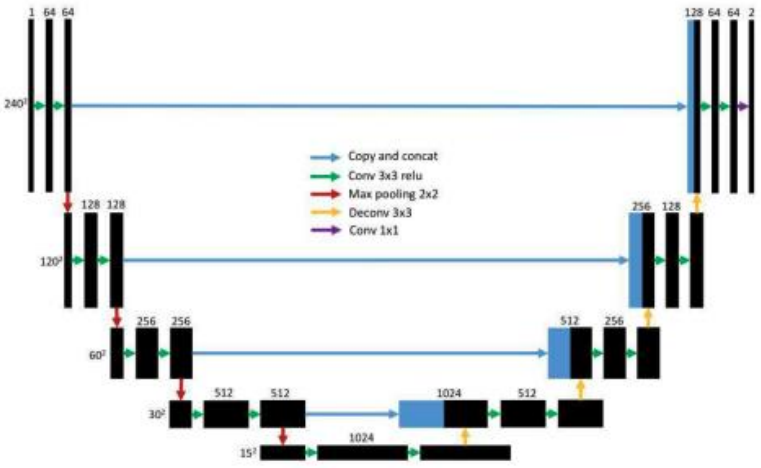


**Figure 19 U-Net [10].**

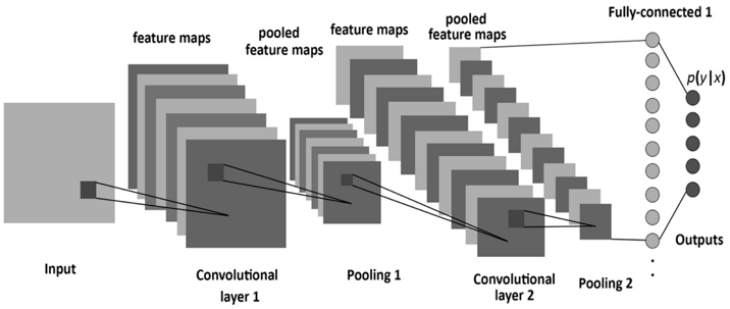


**Figure 20 CNN with two convolutional layers and two pooling layers connected to an ANN [24].**

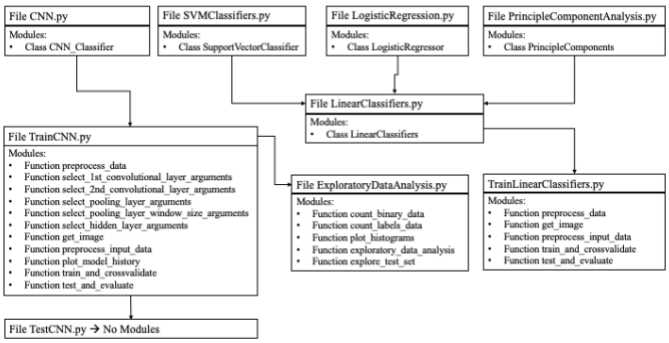


**Figure 21 Implementation's File Organization.**

**Information on the Code:**

- The code used to generate all the figures and experiments is on GitHub at the link: https://github.com/neurogeek953/AMLS_21-22_SN21207190.
- The *dataset.zip* file is not available on GitHub as the file exceeds the file upload memory limit of 25 MBs.
- The code, the training set (*dataset.zip*) and the test set (*test.zip*) are also available on google drive in the file *AMLS_21-22_SN21207190.zip* at the link: https://drive.google.com/file/d/1NikSAw3ReqtoNsZ_tsra_sYxkXrg6GsZ/view?usp=sharing.