Detecting Brain Tumours in Magnetic Resonance Imaging

ECM3420 - Learning from Data

680033128 - 161678

December 8, 2021

I certify that all material in this report which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other university.

1 Introduction and Motivation

Each year, in the UK, more than 11,000 people are diagnosed with a primary brain tumour, of which about half are cancerous, while many others are diagnosed with secondary brain tumours known as metastases [1].

A brain tumour is a mass or growth of abnormal cells in the brain that multiplies in an uncontrollable way.

The distinction between these two lies in whether the tumour originates in the brain (primary) or has spread into the brain from elsewhere (secondary).

Cancerous tumours are also referred to as malignant tumours are more immediately dangerous as they grow at a faster rate and are more likely to persist after treatment. The other kind of tumours are benign or non-cancerous. These grow more slowly and are less likely to return after treatment. They do not spread or invade other parts of the body. Despite this, they are still dangerous and can cause serious harm if they press on vital organs such as the brain [2]. When benign or malignant tumors grow, they cause the pressure inside the skull to increase, leading to brain damage which can be life-threatening.

The risk of developing a brain tumour tends to be greater for adults although they can start to develop at any age. Brain tumours account for 85 to 90 percent of all primary Central Nervous System (CNS) tumours [3].

Surgery is often used to remove brain tumours with the aim being to remove as much abnormal tissue as safely as possible. While treatment for non-cancerous tumours is often successful and a full recovery is possible this still relies on accurate detection.

The fundamental factors that lead me to research this topic are the number of people that are likely to be affected by this type of disease and the serious quality of life implications if left untreated. The 5-year survival rate for people with a cancerous brain or CNS tumor is approximately 34 percent for men and 36 percent for women.

The prognosis for patients having developed a brain tumours can be greatly improved through early detection leading to timely treatment. This reduces the impact of surgery and treatment which is likely to have a positive impact on patients' quality of life, and survival rates. [4].

Diagnosis is difficult so this is an area where we can apply machine learning to make initial evaluations, where ordinarily we would need highly trained specialists to make the determination.

In applications of Machine Learning (ML) and Artificial Intelligence (AI), automated classification techniques have shown a consistently higher degree of accuracy than manual classification. Therefore, by developing a deep learning (DL) based detection and classification system, the performance of brain tumour recognition can be improved. This can take advantage of Convolution Neural Network (CNN) and other types of Artificial Neural Network (ANN) models, along with Transfer Learning (TL) to aid doctors across the world.

2 A - Main Objectives

The application of machine learning in detection brain tumours is likely to significantly reduce the cost and time taken for diagnosis. The sheer complexity of brain tumours makes them hard to detect, attributed to the varying locations and sizes of these abnormalities. This makes it really difficult to build a complete understanding of the nature of the tumor. At present a professional neurologist is required for MRI analysis. In developing countries the application of ML in the field of MRI analysis could go a long way to reduce the strain on healthcare systems, by reducing the reliance on knowledgeable experts. We can use ML to automate the initial stage of detection, before requiring verification from doctors who specialises in the brain and nervous system (neurologists). This

would makes it less challenging and time-consuming to generate reports from MRI scans. Waiting times ahead of patients receiving treatment can therefore be significantly reduced.

This report will focus on interpretation though classification techniques to distinguish between the different types of brain tumours detailed in the dataset. The model will also be able to determine images containing no brain tumour. The two models to be explored in this report are Convolutional Neural Networks (CNNs) and the simpler Feedforward Neural Network (FNN).

The following list several of the subtasks that are to be completed as well as potential snags - all of the below are discussed later in the report:

- Images are too large for efficient training - the reduction in image size does not prove to be a significant hindrance to accuracy.
- Images are taken from different angles - the model should be able to distinguish tumours from multiple orientations
- Images intensities are not bounded - brightness level will be normalized
- Class imbalance - the number of images in each category is roughly equal
- Training examples are insufficient - data augmentation can provide additional examples from which the model can learn
- Images from the two datasets are different sizes - preprecessing stage will crop and standardise the images

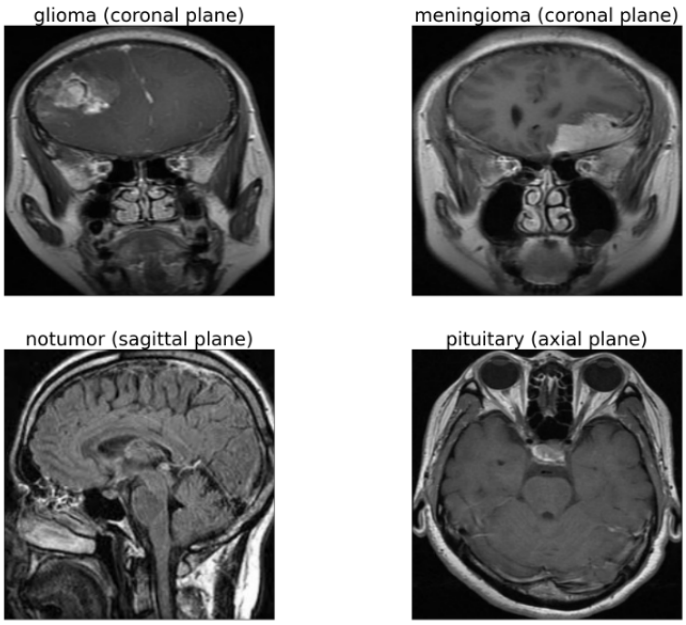
In accordance with the World Health Organization (WHO), the proper diagnosis of a brain tumor involves the detection, identification, and classification of the brain tumor by malignancy, grade, and type. While these models will not be focused on image segmentation, which highlights the exact area of the brain believed to be occupied by the tumour. Further experimentation in diagnosis could provide the means to not only detect and make classifications, but to also grade and identify the location of the tumour. The model could be adapted in future research to incorporate these features provided available datasets contain the ground truth masks required for training the extended models.

3 B - Dataset

The dataset used in this analysis is a combination of the following two datasets: kaggle & figshare.

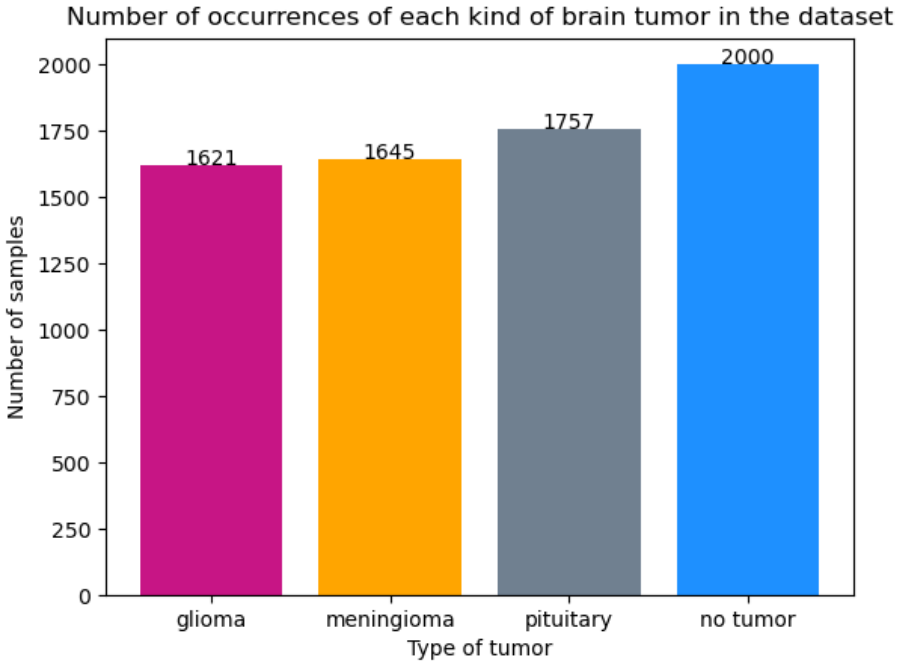
The initial kaggle dataset appears to have the problem that the glioma class of images are not correctly categorised. This is realised from the results of other people's work and as such the images from figshare have been used for this category instead.

Figure 1: Examples of each of the four classifications



The chosen dataset has greater variation than some others covering MRI brain imaging due to the inclusion of scans in all three planes: axial (top-down), sagittal (side profile) and coronal (front-to-back). The pixel intensity and location of these intensities constitute the features of the dataset and allow the model to identify patterns in the images to identify the different parts of the MRI scan such as the skull, brain tissue, gland and of course the tumour itself.

The final dataset contains 7023 images of human brain MRI images which are classified as: glioma, meningioma, pituitary and no tumour shown in the examples in *Figure 1*. The images are already split into training and testing folders and each folder has four more subfolders for the respective tumor classes. The use of data augmentation within the training set should provide a sufficient number of examples off all three MRI planes for each class to be accurately prediction when it comes to testing.



The geographical intensities of the pixels within each of the images constitute input variables and the labels or categories are output variables. The model will look for patterns in the intensities to determine the possible outcomes and assign a probability based on the likelihood that an image belongs to a particular class. The original data consists of black and white 256x256 pixel square images. These have been normalised so that each pixel brightness falls within the range zero to one which is a ratio of the original intensity represented in the original 8-bit greyscale, 256-shade tonal scale.

4 C - Data Exploration - Features Engineering and Data Cleaning

While the original dataset is conveniently organised into training and testing directories with an approximately 80/20 split it lack a validation set. Validation is useful to fine tune higher level hyperparameters and will indicate how well the model is doing on images that are not being used in training. The validation set will give a sense of when the model has reached the best performance possible. After this point we can cease training through a process referred to as early stopping.

From the testing directory a further validation set was created using a 50/50 split. The resulting sets constitute: 5712 training images $\approx 80\%$; 655 test images and 656 validation images $\approx 10\%$ each. To preemptively reduce the effects of overfitting and allow the model to become more generalised and robust, several layers of data augmentation are performed on the training set.

From the previous graph we can see a roughly even split across the different categories. As such the class imbalance problem should be negligible. If the dataset had a very large bias toward a particular category then we could apply focal loss [5] to improve the model accuracy for harder images with less available training examples. Alternatively we could apply greater levels of data augmentation for these classes. This would help reduce misclassification for less frequent kinds of tumours and ensure stable detection across the different types.

Worth noting is that the size of the images in the dataset are different. Therefore a pre-processing stage is included to resize the image to the desired dimensions and remove the extra margins.

5 D - Models and Variations

5.1 Feedforward Neural Network

The first model that I chose to implement consists of a fully connected feedforward neural network (FNN) referred to as a Multi-layer Perceptron (MLP). This consists of a single hidden layer with 80 nodes and an output layer with four nodes corresponding to each of the four categories. For the hidden layer the most suitable activation was found to be the rectified linear activation function (ReLu). An activation function defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. The choice of activation function has a large impact on the capability and performance of the neural network. ReLu is one of the most common functions used for hidden layers owing to both its ease of implementation and effectiveness at overcoming the limitation of some other activation functions. The one notable drawback of the ReLu units are that problems can arise from saturated or "dead" neurons. Though it does address the vanishing gradient problem encountered by the sigmoid function. It does this by assigning an output of zero to all negative inputs. This effectively switches off that neurone so that the model does not use that input to feed the neurones in the next layer. It turns off parts of the network and stops it from picking up any signals coming into that subsequent node. The ReLU function may effectively switch off so many neurones after training the neural network (NN), that when applying the model onto another set of images that may require some of these switched off neurones to actually learn the intricate patterns, the NN is unable to identify the image. It is unlikely that the dying ReLu problem will emerge in the model and as such the model utilises the standard ReLu implementation. If required Leaky ReLu could be used to address the shortcomings of ReLu which uses shallow positive gradients to represent negative values.

The softmax activation function is used for the output to convert the hidden layer into what is essentially a probability distribution of how confident the model is that a particular image belongs to each class.

I had considered expanding the model to include a dropout layer. This is a technique where randomly selected neurones are ignored during training. The contribution of these to selected nodes to the activation of downstream neurones is temporarily removed. This regularisation technique seeks to avoid overfitting (increase the validation accuracy) thus increasing the generalising power of the model. Unfortunately testing with dropout rates at increments of 0.1 between 0.2 and 0.5 saw no significant improvement and in general resulted in nearly identical recall but with lower precision leading to a lower f1-score and weaker performance overall.

The above table displays the performance of the FNN model. The fields are as follows:

- Accuracy - the number of classifications correctly predicted by the model divided by the total number of predictions made.

Table 1: Classification report for FNN model

	precision	**recall**	**f1-score**	**support**
glioma	0.82	0.69	0.75	136
meningioma	0.82	0.81	0.82	156
notumour	0.98	0.99	0.98	212
pituitary	0.82	0.93	0.88	151
accuracy			0.87	655
macro avg	0.86	0.86	0.86	655
weighted avg	0.87	0.87	0.87	655

- Macro Avg - calculates metrics for each label, and finds their unweighted mean. This does not take label imbalance into account.
- Weighted Avg - calculates metrics for each label, and finds their average weighted by support (the number of true instances for each label). This alters macro to account for label imbalance; it can result in an F-score that is not between precision and recall.
- Precision - intuitively it is the ability of the classifier not to label as positive a sample that is negative.
- Recall - intuitively it is the ability of the classifier to find all the positive samples.
- F1-score - interpreted as a weighted harmonic mean of the precision and recall.
- Support - the number of occurrences of each class in the test set.

Precision is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall is defined as follows:

$$Recall\ (TPR) = \frac{TP}{TP + FN} \tag{2}$$

Where TP is the number of true positive, FN the number of false negatives and FP the number of false positives. Additionally recall can be considered as the True Positive Rate (TPR).

We can combine the values for precision and recall to formulate an F1-score (3) which is used to make comparisons between different machine learning models.

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{3}$$

The β parameter allows us to control the trade-off of importance between precision and recall. For $\beta < 1$ we focus more on precision while $\beta > 1$ focuses more on recall.

The F-beta score weights recall more than precision by a factor of β. For our model $\beta = 1$ means recall and precision are equally important.

The simple FNN model detects the occurrence of no brain tumour with 100% accuracy. This would appear to be on the the most important distinctions because for an real world test where the system cannot with confidence say there is no tumour we can then proceed with additional investigation to determine if the tumour is one of the other three that we are able to categorise or something different entirely. According to the normalized confusion matrix the model does appear to be weaker and correctly distinguishing glioma and meningioma tumours. This may be one case for including focal loss in the training stage.

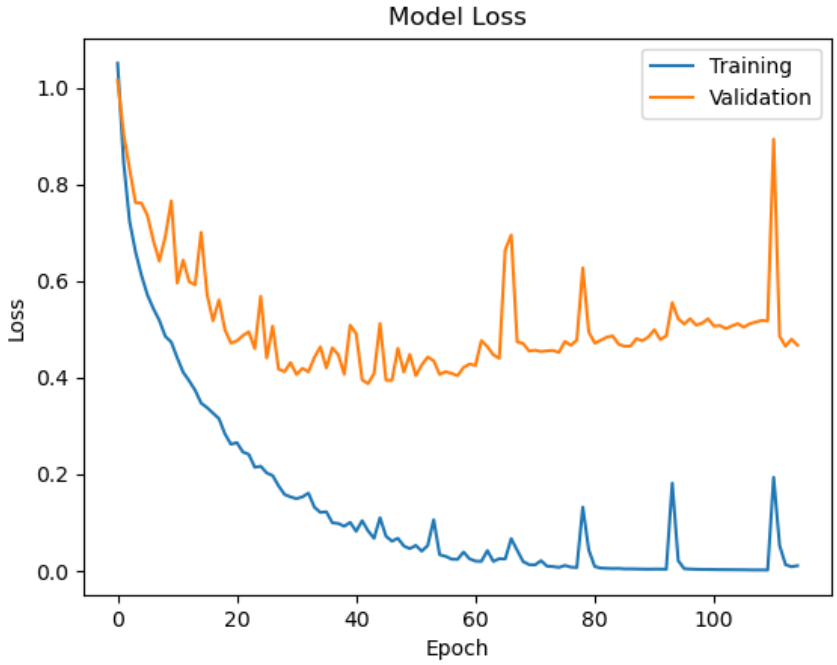
Testing with image size 50, 100, 150 and the original 256 square pixels the was found to be no reduction in accuracy, whereas the smaller images has the added advantage of reducing the training time.

Furthermore, looking at the batch size per epoch having tested at batches of 16, 32, 64, 128 these models were found to perform with higher accuracy where 32 images are processed before each update to the model. With poorer performance observed either side of this optimal.

The following learning curves provides an indication of the model overfitting. This is indicative from:

- The plot of training loss continuing to decrease with experience.
- The plot of validation loss decreasing to a point and beginning to increase again.

In this situation, the inflection point in validation loss may indicate the need to halt training, as experience after that point likely shows the dynamics of overfitting. This appears around 60 epochs as the validation curve reaches the minimum and begins to rise again.



The goal of the learning algorithm is to find a good fit and the second model using CNNs will hope to address the overfitting exhibited by the simpler FNN network.

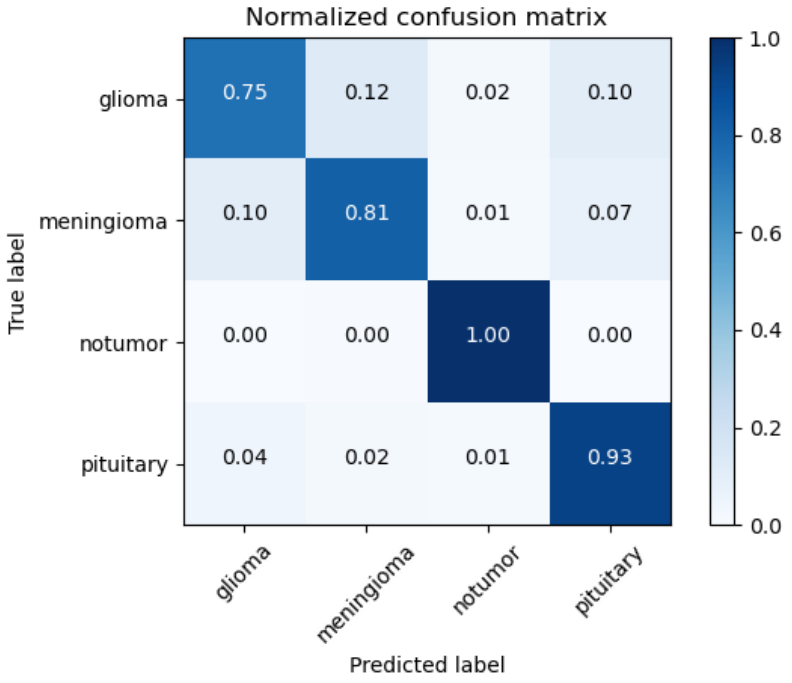
A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

The loss of the model will almost always be lower on the training dataset than the validation dataset. This means that some gap should be expected training and validation loss learning curves, referred to as the generalisation gap.

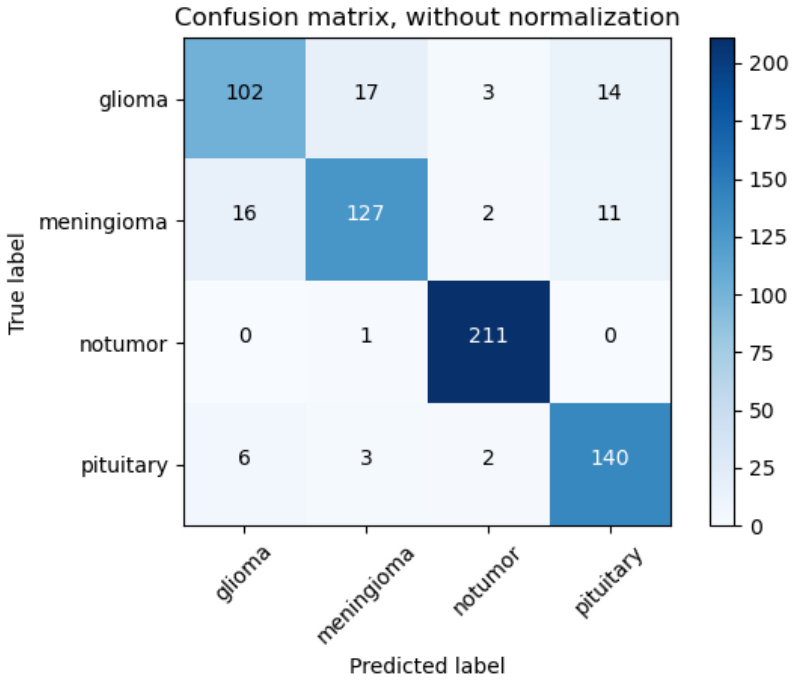
A learning curves plot shows a good fit when:

- The plot of training loss decreases to a point of stability.
- The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

The confusion matrices below provide an intuitive view to evaluate the quality of the output of the classifier on the brain tumour dataset. The diagonal elements represent the number of categorisation for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The colour gradient indicates the proportion of true labels that are predicted to belong to each category. As such we should see that the rows sum to one, forgiving rounding, which appears to be the case. The higher the diagonal values of the confusion matrix the better as this indicates the model has made many correct predictions.



The figures show the confusion matrix with and without normalization (by the number of elements in each class). The advantage of the normalised matrix is that is provides a clearer visual interpretation of which classes are being misclassified as it account for any class imbalance in the data used to make the predictions.



An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots the parameter True Positive Rate (TPR) against the False Positive Rate (FPR).

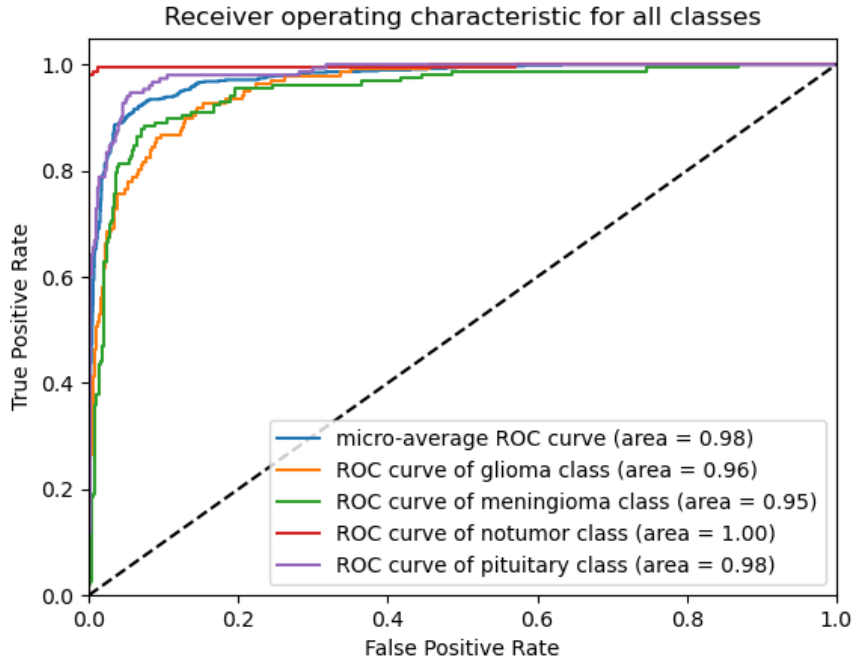
True Positive Rate is a synonym for recall so follows the same definition as previously shown in (2).

False Positive Rate is defined as follows:

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

ROC is a probability curve and the Area Under Curve (AUC) metric represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC the better the model is at telling one class from another.



5.2 Convolutional Neural Network

backpropagation guru dropout

maxpooling2d conv2d

Adding additional units to the convolutional layers or increasing the number of layers resulted in training times that were in the range of several minutes compare to the mere seconds taken to train the FNN model.

6 E - Recommended Model

The recommended model is CNN as this results in higher f1-scores. There may be a slight trade-off with efficiency and training times although this should not be greatly significant.

7 F - Discussion - Summary Key, Findings and Insights

8 G - Future Direction - Possible Flaws and Plan of Action

The capabilities of the model could be expanded to include segmentation masking overlaying the location of the tumour. This would allow doctors and specialist to gain a clearer indication of the size and location of the disease and could potentially be used to direct the best course of treatment. Furthermore the World Health Organisation recognises more than 130 different kinds of brain tumours [6], so the model could be expanded to cover the most serious and most widespread of these.

9 Deliverables

The identification of brain tumor location is also done using a CNN-based model by segmenting the brain tumor.

10 Conclusion

Fix hyperlinks with visited timestamp discussion of the effect of reducing image size in improving training times description of loss and accuracy metrics confusion matrix roc curve

References

[1] "Brain tumours — NHS," 2020. [Online]. Available: https://www.nhs.uk/conditions/brain-tumours/ [Accessed:23-Nov-2021]

[2] "Benign brain tumours — NHS," 2020. [Online]. Available: https://www.nhs.uk/conditions/benign-brain-tumour/[Accessed:23-Nov-2021]

[3] "Brain Tumour: Statistics — Cancer.Net," 2021. [Online]. Available: https://www.cancer.net/cancer-types/brain-tumor/statistics[Accessed:23-Nov-2021]

[4] L. Lansdowne, "Early Detection of Brain Tumours and Beyond — Technology Networks," 2021. [Online]. Available: https://www.technologynetworks.com/cancer-research/blog/early-detection-of-brain-tumors-and-beyond-354432[Accessed:23-Nov-2021]

[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[6] "Adult brain tumour types — The Brain Tumour Charity," 2021. [Online]. Available: https://www.thebraintumourcharity.org/brain-tumour-diagnosis-treatment/types-of-brain-tumour-adult/\protect\protect\leavevmode@ifvmode\kern+.2222em\relax[Accessed:23-Nov-2021]

```
from keras.callbacks import
EarlyStopping,ReduceLROnPlateau,ModelCheckpoint,TensorBoard,LambdaCallback
from keras.layers import Input,Dropout, Dense,GlobalAveragePooling2D
from keras.models import Sequential,Model
from keras.applications.resnet import ResNet50
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,
plot_confusion_matrix, \
    accuracy_score, roc_curve, auc
from tensorflow.python.ops.numpy_ops import np_config
np_config.enable_numpy_behavior() # used for ravel function to compress one
hot encoding
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from tensorflow import keras
import tensorflow as tf
from tqdm import tqdm
import seaborn as sns
import numpy as np
import itertools
import datetime

import cv2
import os
import io

labels = ['glioma', 'meningioma', 'notumor', 'pituitary']
plane = ['coronal', 'coronal', 'sagittal', 'axial']

x_train = [] # training images.
y_train = [] # training labels.
x_test = [] # testing images.
y_test = [] # testing labels.

image_size = 50

train_path = 'cleaned/Training'
test_path = 'cleaned/Testing'

for label in labels:
    train_dir = os.path.join(train_path, label)
    for file in tqdm(os.listdir(train_dir)):
        image = cv2.imread(os.path.join(train_dir, file), 0)  # load images
in gray.
        image = cv2.bilateralFilter(image, 2, 50, 50)  # remove images
noise.
        image = cv2.resize(image, (image_size, image_size))  # resize
images.
        image = image[:, :, np.newaxis] # adds the value one representing
greyscale
        x_train.append(image)
        y_train.append(labels.index(label))

    test_dir = os.path.join(test_path, label)
    for file in tqdm(os.listdir(test_dir)):
        image = cv2.imread(os.path.join(test_dir, file), 0)
        image = cv2.bilateralFilter(image, 2, 50, 50)
        image = cv2.resize(image, (image_size, image_size))
        image = image[:, :, np.newaxis]
        x_test.append(image)
```

```
        y_test.append(labels.index(label))

x_test, x_val, y_test, y_val = train_test_split(x_test, y_test,
test_size=0.5, random_state=123) # testing dir split into test and
validation sets
# research EarlyStopping

x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)
x_val = np.array(x_val)
y_val = np.array(y_val)

# 7023 images total
print(x_train.shape) # 81% 5712 images
print(x_test.shape) # 9% 655 images
print(x_val.shape) # 9% 656 images


# images = [x_train[i] for i in range(15)]
# fig, axes = plt.subplots(3, 5, figsize = (10, 10))
# axes = axes.flatten()
# for img, ax in zip(images, axes):
#     ax.imshow(img)
# plt.tight_layout()
# plt.show()

# Data augmentation
# ImageDataGenerator transforms each image in the batch by a series of
random translations, rotations, etc.
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True)

# After you have created and configured your ImageDataGenerator, you must
fit it on your data.
datagen.fit(x_train)


plt.figure(figsize=(20, 16))

images_path = ['/glioma/Tr-glTr_0000.jpg', '/meningioma/Tr-meTr_0000.jpg',
'/notumor/Tr-no_0060.jpg', '/pituitary/Tr-piTr_0000.jpg']

for i in range(4):
    ax = plt.subplot(2, 2, i + 1)
    img = cv2.imread(train_path + images_path[i])
    plt.imshow(img)
    plt.title(labels[i] + ' (' + plane[i] + ' plane)', fontsize=28)
    plt.xticks([], minor=True)
    plt.yticks([], minor=True)
plt.show()


# This callback will stop the training when there is no improvement in the
loss for three consecutive epochs.
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10)
```

```
# Convolutional Neural Network Model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(image_size, image_size, 1)), # B&W images
    tf.keras.layers.Rescaling(1. / 255, input_shape=(image_size,
image_size, 1)), # normalize Images into range 0 to 1.

    # # Convolutional layer 1
    # tf.keras.layers.Conv2D(16, (3,3), input_shape=(image_size,
image_size, 1), activation='relu'),
    #
    # tf.keras.layers.MaxPooling2D(),
    # tf.keras.layers.Flatten(),
    #
    # tf.keras.layers.Dense(units=80, activation='relu'),
    # tf.keras.layers.Dropout(0.2),
    # tf.keras.layers.Dense(units=len(labels), activation='softmax'),
    #

    # tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
    # tf.keras.layers.MaxPooling2D(),
    # # tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    # # tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(labels), activation='softmax')
])



# Possible Custom CNN
# model_2.add(Conv2D(16,(3,3),padding='same',input_shape
=X_train_prep[0].shape))
# model_2.add(BatchNormalization())
# model_2.add(Activation('relu'))
# model_2.add(MaxPooling2D(pool_size=(2,2),strides=2,padding = 'same'))
#
# model_2.add(Conv2D(32,(3,3),padding='same'))
# model_2.add(BatchNormalization())
# model_2.add(Activation('relu'))
# model_2.add(MaxPooling2D(pool_size=(2,2),strides=2,padding = 'same'))
#
# model_2.add(Flatten())
# model_2.add(Dense(nClasses,activation='sigmoid'))



# Feedforward Neural Network Model
# model = tf.keras.Sequential([
#     tf.keras.layers.Input(shape=(image_size, image_size, 1)),  # B&W
images
#     tf.keras.layers.Rescaling(1. / 255, input_shape=(image_size,
image_size, 1)), # normalize Images into range 0 to 1.
#     tf.keras.layers.Flatten(),
#     tf.keras.layers.Dense(80, activation='relu'),
#     # tf.keras.layers.Dropout(0.2), # no significant gain in performance
observed
#     tf.keras.layers.Dense(len(labels), activation='softmax'), # total
probability to be one but not necessary for hidden layers
```

```
# ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

print(x_train.shape, y_train.shape)
train_ds = tf.data.Dataset.from_tensor_slices((x_train,
y_train)).shuffle(5000, reshuffle_each_iteration=True).batch(32)
# training the model
history = model.fit(train_ds,
                    epochs = 250,
                    verbose = 1,
                    validation_data = (x_val, y_val),
                    callbacks=[callback])



print(history.history.keys())

# plotting losses
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()

#  plotting accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper left')
plt.show()


# validate on val set
predicted_prob = model.predict(x_test)
predicted_class = np.argmax(predicted_prob, axis=-1)

# Generate confusion matrix
def plt_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
```

```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    max_range = round(np.max(cm))
    plt.clim(0, max_range)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout(pad=2)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, predicted_class)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plt_confusion_matrix(cnf_matrix, classes=labels,
                     title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plt_confusion_matrix(cnf_matrix, classes=labels, normalize=True,
                     title='Normalized confusion matrix')
plt.show()



# Generating ROC figures

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
y_test_one_hot = tf.one_hot(y_test, 4) # depth 4

for i in range(len(labels)):
    fpr[i], tpr[i], _ = roc_curve(y_test_one_hot[:, i], predicted_prob[:,
i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# print(y_test_one_hot)
# print(predicted_prob)

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_one_hot.ravel(),
predicted_prob.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]))
```

```
for i in range(len(labels)):
    plt.plot(fpr[i], tpr[i], label='ROC curve of ' + labels[i] + ' class
(area = {1:0.2f})'
                                   ''.format(i, roc_auc[i]))

# Plot of a ROC curve for multiple classes
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for all classes')
plt.legend(loc="lower right")
plt.show()


# # Plot of a ROC curve for a specific class
# chosen_class = 1 # meningioma
# plt.figure()
# plt.plot(fpr[chosen_class], tpr[chosen_class], label='ROC curve of ' +
labels[chosen_class] + ' class (area = %0.2f)' % roc_auc[chosen_class])
# plt.plot([0, 1], [0, 1], 'k--')
# plt.xlim([0.0, 1.0])
# plt.ylim([0.0, 1.05])
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive Rate')
# plt.title('Receiver operating characteristic of ' + labels[chosen_class]
+ ' class')
# plt.legend(loc="lower right")
# plt.show()


# classification report
print(classification_report(y_test, predicted_class, target_names=labels))
```