BRAIN TUMOUR RECOGNITION: PROJECT IMPLEMENTATION

Report prepared by:

Franciszek Liszka, 24856700
Raeyen Nuttall, 24554077
Sandul Mapalagama, 25002039
Wendy Lam, 13582895
Yang Chang, 14075218
Yiran Guo, 14068516

31256 Image Processing and Pattern Recognition

TABLE OF CONTENTS

1. INTRODUCTION

Technology is increasingly being used to address complex healthcare challenges within the field of medical diagnostics. With diagnostic imaging, developments in image processing and pattern recognition present opportunities to facilitate more efficient and accurate detection of a wide range of clinical conditions, including brain tumours.

Diagnosing brain tumours presents challenges for medical professionals due to the variety of possible shapes, sizes and locations (Abdusalomov et al., 2023). While magnetic resonance imaging (MRI) can produce highly detailed images of the brain, detecting tumours within these images is a difficult and manual process that is dependent on the interpretation of the medical professional overseeing the case (Amin et al., 2022). As such, the application of image processing and pattern recognition techniques can assist in refining the interpretation of MRI scans thus providing insights to medical professionals that may be missed through manual identification. This will ensure that patients can access effective treatment planning.

This report will outline the implementation of our project to develop a robust system that is capable of accurately detecting and classifying brain tumours from MRI scans. In Section 2, we highlight the need for efficient diagnosis of brain tumours and how our project aims to contribute to this area. In Section 3, we describe the code produced to implement the project and the details of our development process. In Section 4, we outline our project management strategy. In Section 5, we demonstrate the results obtained from our data and models. In Section 6, we discuss our learnings and challenges for this project.

2. PROBLEM ANALYSIS

2.1. Problem summary

A brain tumour is characterised by the development of a mass of abnormal cells in the brain (Amin et al., 2019). Brain tumours can be benign or malignant, and can develop in the brain or spread to the brain from elsewhere in the body (Sapra et al., 2013). The brain tumours most frequently seen in medical practice are meningioma, glioma and pituitary tumours, shown in Figure 1 (Abdusalomov et al., 2023).

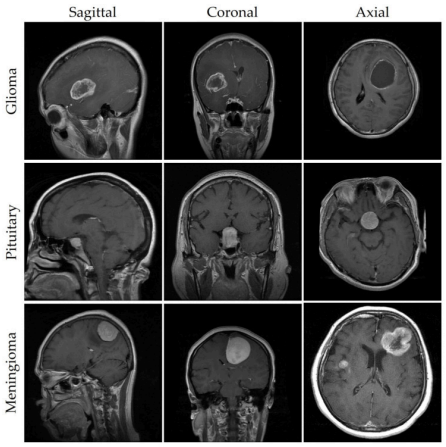

Figure 1. Glioma, pituitary and meningioma tumours in the brain (Abdusalomov et al., 2023)

Brain tumours, irrespective of whether they are benign or malignant, constitute a major health risk. These tumours can severely impair biological functions, create permanent brain damage and can be life-threatening (Sapra et al., 2013). Therefore, quick and accurate diagnosis of a brain tumour is critical in improving a patient's prognosis and ensuring effective treatment planning.

For early detection of abnormalities in the brain, MRI is the preferred imaging technique as it allows for in-depth examination of the skull and brain through providing high contrast axial, coronal and sagittal images (Abdusalomov et al., 2023). Medical professionals must rely on their interpretation of these images in order to detect and classify tumours. However, this can often be subjective depending on the quality of the image and the level of experience the practitioner has in identifying brain tumours (Abdusalomov et al., 2023). Given these complexities, accurate diagnosis of a brain tumour by human visual examination alone is a challenging and error-prone process (Abdusalomov et al., 2023).

2.2. Project outline

In this project, we aim to address the problem outlined by leveraging image processing and pattern recognition techniques to produce automated detection and classification of brain tumours from MRI scans. By creating a more efficient process for the preliminary analysis of a brain image, medical professionals will be better able to make decisions for their patients. Medical professionals will be able to use the results from our model to accurately determine what measures are needed to further substantiate the diagnosis, such as performing a biopsy, and what treatment options will be most suitable for the patient.

The data we used to develop our project is the publicly available 'Brain Tumour MRI Dataset', which contains 7022 images of MRI brain scans that are split into four classes (Nickparvar, 2021). The classes are: glioma (1621 images), meningioma (1645 images), pituitary (1757 images) and no tumour (2000 images). The dataset is a combination of three widely downloaded and cited datasets: figshare, SARTAJ and Br35H, and has been validated by numerous studies (see Abdusalomov et al., 2023). In order to improve the generalisability of the model, we used data augmentation strategies to diversify our training samples.

2.3. Proposed solutions and evaluation methodologies

In our 'Project Requirements and Specifications Report,' we identified that convolutional neural networks (CNNs) would be particularly useful for our project due to its ability to recognise patterns in images. CNNs are a multilayer neural network consisting of a convolutional layer, pooling layer and fully connected layer. Input data moves through convolution layers, which detect features through the application of filters. Pooling layers then reduce the spatial dimensions of the images to reduce the complexity of calculations. Fully connected layers then output the final predictions.

There are different architectures based on CNNs that we discussed in terms of their theoretical applications to our project. After careful consideration of the requirements of our project, we selected the following neural network architectures to explore for our multi-class classification problem: CNN, ResNet, VGG and Inception. Our reasons for selecting these architectures are outlined in the table below.

Architecture	Reason for selection
Convolutional Neural Network (CNN)	• **Simplicity and efficiency:** The simple and straightforward architecture of CNNs (as outlined above) facilitates rapid implementation and training. • **Feature extraction:** CNNs excel at extracting a wide array of features, such as edges, textures and patterns. • **Strong baseline:** Initial experiments can be conducted with a basic CNN architecture, allowing for progressive exploration of more complex models when necessary.
Residual Network (ResNet)	• **Deep architecture:** This enables the capture of highly intricate and hierarchical features in images, and is valuable when working with datasets that are complex • **Vanishing gradient:** Residual connections addresses the vanishing gradient issue meaning that training deep networks does not result in a loss of information • **High performance:** It has consistently demonstrated high performance across diverse computer vision tasks
Visual Geometry Group (VGG)	• **Uniform architecture:** Convolutional layers in VGG share the same filter size (3x3) and are consistently followed by max-pooling layers. This uniformity simplifies model implementation and comprehension • **High performance:** It has consistently demonstrated high performance in image classification tasks • **Versatility:** It is highly adaptable and can be tailored to different tasks as it can handle a wide range of image data
Inception	• **Multiple pathways:** It utilises multiple convolutional filter sizes within the same layer, enabling it to identify a wide range of complex features concurrently • **Dimension reduction:** It incorporates dimension-reduction techniques (e.g. 1x1 convolutions) to ease computation burden and preserve information, which is important when working with large datasets • **High performance:** It has consistently demonstrated high performance in computer vision tasks, particularly where the dataset is significantly diverse

To evaluate the performance of our model in classifying images into four distinct classes (glioma, meningioma, pituitary and no tumour), we use a number of different metrics to help us identify strengths and areas for improvement. Accuracy and loss are used to measure the proportion of predictions the model has correctly identified relative to the dataset, and the difference between the model's predicted outputs and actual labels. Precision and recall are used to determine the reliability of positive predictions for each class and the model's ability to identify all instances of a class, resulting in an F1-score (Sokolova & Lapalme, 2009). A confusion matrix is used to recognise errors, particularly false positives and false negatives (Provost & Fawcett, 2014). Real-time inference assesses how rapidly the model can provide outcomes. Cross-validation ensures the generalisability of the model. Given the need for an accurate model in determining the presence of a brain tumour in an MRI scan, these metrics are critical to the success of the overall project.

3. CODE IMPLEMENTATION

3.1. Description of the code

This section outlines the code implementation for brain tumour classification for four models: CNN, VGG, ResNet and Inception. All models have been built using Python.

3.1.1. Data preprocessing and analysis

Before building our models, we engaged in data preprocessing and analysis to help us understand our dataset. This also allowed us to transform the data in a manner that would increase the generalisability of our models.

Visualising class distribution

Visualising the class distribution in the dataset allowed us to identify whether there was an imbalance in class distribution. This is important as imbalanced datasets can lead to bias when training the models for classes that have more samples. Understanding the imbalance in our dataset allowed us to select the appropriate models and evaluation metrics. It also informed our approach to data augmentation and stratified sampling in data splits.

To visualise the class distribution, we performed a count of images in each class. This was then displayed on a bar plot, as seen in Figure 2.

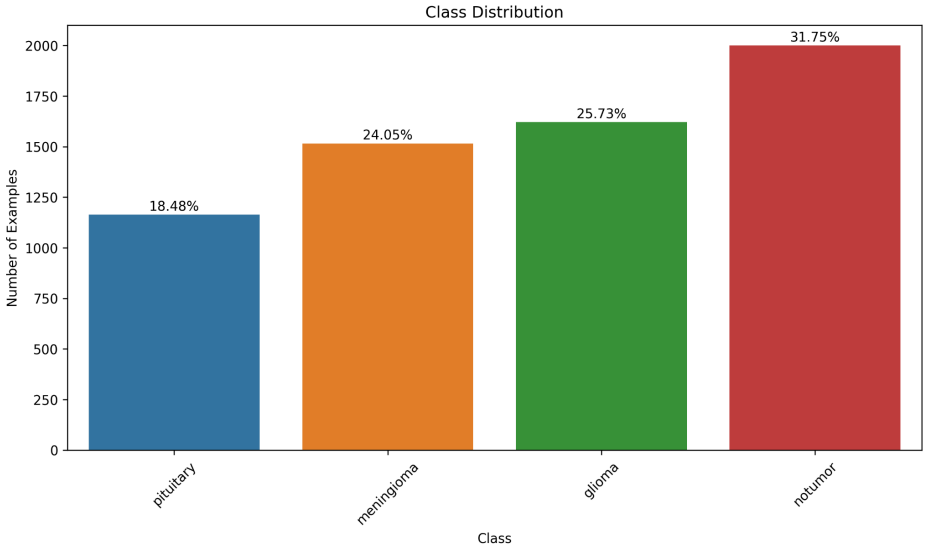

Figure 2. Class distribution graph

Loading and resizing images

In the code shown below in Figure 3, we used the get_image_data function to load and resize our images. Using the OpenCV library (cv2), the cv2.imread(path) function loads an image from a designated file path and the cv2.resize function transforms the image in accordance with the dimensions specified in the width and height parameters. The resize function also ensured uniformity in the dimensions of all images in the dataset.

We then used the load_dataset function, which serves as a data aggregator, to collect images and their respective labels. The for loop systematically traverses the folders corresponding to different labels. Each image is then loaded, resized and paired with the relevant label. The load_dataset function ultimately returns three crucial components: a NumPy array housing the image data, another NumPy array containing the corresponding labels, and the label_map dictionary. The label_map dictionary performs the function of mapping label names to their respective numeric indices. These components collectively form the foundation upon which subsequent model training and evaluation processes will be built.

```python
def get_image_data(path, width, height):
    img = cv2.imread(path)  # load
    return cv2.resize(img, (width, height))  # resize


def load_dataset(IMG_WIDTH=225, IMG_HEIGHT=225):
    base_path = "data"
    images, labels = [], []
    label_names = os.listdir(base_path)
    label_map = {name: idx for idx, name in enumerate(label_names)}

    for label_name, label_idx in label_map.items():
        folder_path = os.path.join(base_path, label_name)
        for filename in os.listdir(folder_path):
            img_path = os.path.join(folder_path, filename)
            img = get_image_data(img_path, IMG_WIDTH, IMG_HEIGHT)
            images.append(img)
            labels.append(label_idx)

    return np.array(images), np.array(labels), label_map
```

Figure 3. Loading and resizing images

Data augmentation

We aimed to enhance the dataset by introducing augment versions of existing images. These augmented images will serve to expose the model to a broader spectrum of image variations, ultimately contributing to its robustness and performance.

In the code shown below in Figure 4, we start by defining datagen, an instance of the ImageDataGenerator class. This class is part of the Keras library and has a rich set of tools for transforming images. To configure the augmentation strategy, the code specifies a range of transformations, including:

- Rotation, with a range of up to 5 degrees.
- Width shift, with a range of up to 5% of the image's total width.
- Height shift, with a range of up to 5% of the image's total height.
- Zooming, with a range of up to 10%.

For every input image, two additional augmented images are generated by temporarily expanding the dimensions of the input image to satisfy the requirements of the ImageDataGenerator, then applying predefined transformations to create a new image. We then have to ensure that the pixel values of the augmented image are correct. The final output of the augment_data function is a pair of NumPy arrays: augmented_images and augmented_labels. These arrays collectively constitute the augmented dataset and are pivotal for training and evaluating the model.

```python
def augment_data(images, labels, augmentations=2):
    datagen = ImageDataGenerator(
        rotation_range=5,
        width_shift_range=0.05,
        height_shift_range=0.05,
        zoom_range=0.1,
        fill_mode="nearest",
    )

    augmented_images, augmented_labels = [], []
    for img, label in zip(images, labels):
        augmented_images.append(img)
        augmented_labels.append(label)
        for _ in range(augmentations):
            augmented = datagen.flow(np.expand_dims(img, 0), batch_size=1).next()[0]
            augmented_images.append(np.squeeze(augmented).astype(np.uint8))
            augmented_labels.append(label)

    return np.array(augmented_images), np.array(augmented_labels)
```

Figure 4. Data augmentation

Split and augment dataset

The **split_and_augment_dataset** function, as seen in Figure 5 below, splits the dataset, handles the creation of evaluation sets, and applies data augmentation to the training set.

With data splitting, the function operates in two scenarios: with evaluation set (eval_set=True) or without evaluation set (eval_set=False). In the first scenario, the data is divided into two segments – combined training and evaluation (80%) and testing (20%), In the combined training and evaluation segment, the data is split into training (75%) and evaluation (25%). In the second scenario, the data is split into two segments – training (80%) and testing (20%). In both scenarios, stratified sampling is used to ensure an even distribution of classes. The data augmentation process then applies exclusively to the training set to introduce variations to the original images. The function also provides an optional random_state parameter, which allows for reproducible data splitting and augmentation to ensure consistency across different runs.

```python
def split_and_augment_dataset(images, labels, eval_set=False, random_state=None):
    if eval_set:
        # Split into train+eval and test
        (
            train_eval_images,
            test_images,
            train_eval_labels,
            test_labels,
        ) = train_test_split(
            images, labels, test_size=0.2, random_state=random_state, stratify=labels
        )
        # Split train+eval into train and eval
        train_images, eval_images, train_labels, eval_labels = train_test_split(
            train_eval_images,
            train_eval_labels,
            test_size=0.25,
            random_state=random_state,
            stratify=train_eval_labels,
        )
        # Augment only the training data
        augmented_train_images, augmented_train_labels = augment_data(
            train_images, train_labels
        )
        return (
            augmented_train_images,
            augmented_train_labels,
            eval_images,
            eval_labels,
            test_images,
            test_labels,
        )
    else:
        # Split into train and test
        train_images, test_images, train_labels, test_labels = train_test_split(
            images, labels, test_size=0.2, random_state=random_state, stratify=labels
        )
        # Augment only the training data
        augmented_train_images, augmented_train_labels = augment_data(
            train_images, train_labels
        )
        return augmented_train_images, augmented_train_labels, test_images, test_labels
```

Figure 5. Split and augment dataset

3.1.2. Model 1: CNN

This section outlines our approach to building two CNN models that one more complex than the other for brain tumour classification.

CNN 1

As seen in Figure 6, we started by creating a Sequential mode, which is a linear stacked neural network that allows the construction of deep neural networks by adding different layers to the model one-by-one. A convolutional layer is then constructed. It has 32 different convolutional kernels that are 3x3 in size, and an activation function using the Rectified Linear Unit (ReLU) in order to introduce non-linearity. Following that, a maximum pooling layer is added. Maximum

pooling is used to reduce the spatial dimension of the feature map. We use a pooling window of 2x2 to halve the size of the feature map. To connect the convolved and pooled feature maps to the fully connected layer, a spreading layer is added to convert the multidimensional feature maps to one-dimensional vectors. The process between the convolutional layer, pooling window and spreading layer is repeated twice more with increasing numbers of convolutional kernels (e.g. 64, 128). In addition to this, a fully connected layer with 256 neurons is added with the ReLU activation function. Finally, a fully connected layer is added as an output layer with the number of neurons equal to the number of categories in the problem, 4 categories, and softmax is used to produce multi-class classification results

```python
def CNN_1(input_shape, num_classes):
    model = Sequential()

    model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

Figure 6. CNN 1

CNN 2

This model mirrors the structure of CNN 1. As seen in Figure 7 below, the main differences are that CNN 2 does not have a third convolutional layer, and that the fully connected layer has 128 neurons instead of 256.

```python
def CNN_2(input_shape, num_classes):
    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

Figure 7. CNN 2

3.1.3. Model 2: VGG

This section outlines our approach to building a VGG-Net architecture for brain tumour classification, as outlined in Figure 8.

```
1  # VGG-Net
2  model = Sequential()
3  model.add(Conv2D(64, (3, 3), strides=(1, 1), input_shape=input_shape, padding='same', activation='relu', kernel_initializer='uniform'))
4  model.add(Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
5  model.add(MaxPooling2D(pool_size=(2, 2)))
6  model.add(Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
7  model.add(Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
8  model.add(MaxPooling2D(pool_size=(2, 2)))
9  model.add(Conv2D(256, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
10 model.add(Conv2D(256, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
11 model.add(MaxPooling2D(pool_size=(2, 2)))
12 model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
13 model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
14 model.add(MaxPooling2D(pool_size=(2, 2)))
15 model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
16 model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel_initializer='uniform'))
17 model.add(MaxPooling2D(pool_size=(2, 2)))
18 model.add(Flatten())
19 model.add(Dense(4096, activation='relu'))
20 model.add(Dropout(0.5))
21 model.add(Dense(4096, activation='relu'))
22 model.add(Dropout(0.5))
23 model.add(Dense(4, activation='softmax'))
```

Figure 8. VGG

The first step is to build the neural network by creating the Sequential model, a linear stack of layers. When the first convolutional layer is added, the shape and channel information of the input data is defined for the input layer model of the architecture, which is an RGB (3-channel) image. In addition, a convolutional layer with 64 convolution kernels and a filter size of 3×3 is created to extract the image features. Each convolutional operation shifts one pixel, and the "same" parameter is used to specify that the size of the output feature maps matches the size of the input feature maps. The ReLU activation function is used to introduce nonlinearities to help the network to learn better. The ‘uniform’ kernel initialiser means that the weights of the convolution kernel are initialised randomly using a uniform distribution. The maximum pooling layer and 2×2 pooling window are used in constructing the pooling layer to reduce the size of the feature map.

The subsequent steps were a series of adding convolutional layers and maximum pooling layers to build the deep VGG architecture. In this process, there is a maximum pooling layer between every two convolutional layers to reduce the size of the feature map. This model structure is repeated several times to extract the features of the image gradually. In each layer, the ReLU activation function is used to non-linearise the features. A Flatten layer was added to flatten the output of the convolutional layer into a one-dimensional vector until the fully connected layer was added at line 18 of the code, as shown in Figure 8. The fully connected layer was then added containing 4096 neurons with the ReLU activation function andDropout layers interspersed between the fully connected layers to prevent overfitting.

Finally, the output layer is defined. Since this project requires multi-classification, the number of output neurons is set to 4. The "softmax" activation function is used to convert the output of the network into the form of a probability distribution of the categories, which is convenient for the subsequent classification decision.

3.1.4. Model 3: ResNet

This section outlines our approach to building ResNet-18 architecture for brain tumour classification. It contains custom auxiliary functions "conv2d_bn", "shortcut", "basic_block" and "residual_block" which are used to build the core components of the ResNet model, including convolutional layers, residual blocks, and auxiliary connections.

The first function as shown in figure 9 "conv2d_bn" is used to create the convolutional layer, which contains the normalisation and ReLU activation functions.

```
def conv2d_bn(x, nb_filter, kernel_size, strides=(1, 1), padding='same'):
    """
    conv2d -> batch normalization -> relu activation
    """
    x = Conv2D(nb_filter, kernel_size=kernel_size,
               strides=strides,
               padding=padding,
               kernel_regularizer=regularizers.l2(0.0001))(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
```

Figure 9. ResNet conv2d_bn

The "shortcut" function (see Figure 10) is then defined to construct a shortcut connection of the residual blocks. The function first gets the shape information of the input and output, and then calculates the scale difference between the two, including the scaling factors of height and width and whether the number of channels is equal. If the scales are different or the number of channels is different, the function uses a 1x1 convolutional layer to adjust the size of the input to match the size of the output. Finally, the function adds the adjusted inputs to the outputs of the residual blocks, achieving residual connectivity, which is one of the key features of ResNet.

```
def shortcut(input, residual):
    """
    shortcut connection, which is the identity mapping part.
    """

    input_shape = K.int_shape(input)
    residual_shape = K.int_shape(residual)
    stride_height = int(round(input_shape[1] / residual_shape[1]))
    stride_width = int(round(input_shape[2] / residual_shape[2]))
    equal_channels = input_shape[3] == residual_shape[3]

    identity = input
# If the dimensions are different, adjust using 1x1 convolution
    if stride_width > 1 or stride_height > 1 or not equal_channels:
        identity = Conv2D(filters=residual_shape[3],
                          kernel_size=(1, 1),
                          strides=(stride_width, stride_height),
                          padding="valid",
                          kernel_regularizer=regularizers.l2(0.0001))(input)

    return add([identity, residual])

```

Figure 10. ResNet shortcut

As shown in Figure 11, the "basic_block" function is used to build the basic residual block in ResNet, which applies two consecutive convolutional layers, each of which is created by the conv2d_bn function. The residuals are then connected by the shortcut function. The 'nb_filter' parameter is the number of convolution kernels, and strides is the step size of the convolution operation.

```
62
63  def basic_block(nb_filter, strides=(1, 1)):
64      """
65     Basic ResNet building block for RESNET-18 and RESNET-34.
66      """
67
68      def f(input):
69          conv1 = conv2d_bn(input, nb_filter, kernel_size=(3, 3), strides=strides)
70          residual = conv2d_bn(conv1, nb_filter, kernel_size=(3, 3))
71
72          return shortcut(input, residual)
73
74      return f
75
```

Figure 11. ResNet basic_block

The "residual_block" function according to Figure 12 is used to build a residual module for each layer. Each residual module is built in a loop, and each module contains several basic residual blocks, which can be repeated several times in each module.

```
77  def residual_block(nb_filter, repetitions, is_first_layer=False):
78      """
79    residual module of each layer is constructed, corresponding to the conv2__g_t in the parameter statistics table of the paper. conv5_x
80      """
81
82      def f(input):
83          for i in range(repetitions):
84              strides = (1, 1)
85              if i == 0 and not is_first_layer:
86                  strides = (2, 2)
87              input = basic_block(nb_filter, strides)(input)
88          return input
89
90      return f
```

Figure 12. ResNet residual_block

In the end, the core architecture of the ResNet model is formed by combining these functions, seen in Figure 13, which are called separately. An input layer is created to define the desired input shape of the model. The first convolutional layer is constructed to identify the image features by calling the constructed "conv2d_bn" function. The first pooling layer is created to downsample the image by calling "MaxPooling2D". Subsequently, four residual blocks, each containing a different number of convolutional and pooling layers, were constructed by calling the "residual_block" function. These residual blocks formed the structure of ResNet-18. Finally, the features are pooled through the global average pooling layer "GlobalAvgPool2D." The category scores are output through the fully connected layer, then "Dense" and the "softmax" activation function is applied to obtain the classification probabilities. The output of the model is stored in "output_".

```python
def ResNet18(input_shape=(115, 115, 3), nclass=4):
    """
    build resnet-18 model using keras with TensorFlow backend.
    :param input_shape: input shape of network, default as (115,115,3)
    :param nclass: numbers of class(output shape of network), default as 4
    :return: resnet-18 model
    """
    input_ = Input(shape=input_shape)

    conv1 = conv2d_bn(input_, 64, kernel_size=(7, 7), strides=(2, 2))
    pool1 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(conv1)

    conv2 = residual_block(64, 2, is_first_layer=True)(pool1)
    conv3 = residual_block(128, 2, is_first_layer=True)(conv2)
    conv4 = residual_block(256, 2, is_first_layer=True)(conv3)
    conv5 = residual_block(512, 2, is_first_layer=True)(conv4)

    pool2 = GlobalAvgPool2D()(conv5)
    output_ = Dense(nclass, activation='softmax')(pool2)

    model = Model(inputs=input_, outputs=output_)
    # model.summary()

    return model

```

Figure 13. ResNet architecture

3.1.5. Model 4: Inception

This section outlines our approach to building an Inception model for brain tumour classification.

```python
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential

IMAGE_SIZE = 115  # Adjust as needed

base_model = InceptionV3(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), include_top=False, weights='imagenet')


# Number of layers to freeze from the beginning
num_layers_to_freeze = 50  # Adjust this value as needed

# Set the first 'num_layers_to_freeze' layers to non-trainable
for layer in base_model.layers[:num_layers_to_freeze]:
    layer.trainable = False
from tensorflow.keras.layers import Flatten, Dense, Dropout

model = Sequential()
model.add(Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
model.add(base_model)
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))

# Adjust the number of units in the output layer based on your dataset
model.add(Dense(len(label_map), activation='softmax'))
```

Figure 14. Inception

As seen in Figure 14 above, the Inception model has been structured as follows. The code initiates the model by defining an input layer suited for RGB images (lines 5-6). It is tailored to accommodate images sized at 115x115 pixels, specifying the input shape and channel information. The InceptionV3 pre-trained architecture in lines 9-11 is equipped with weights from the ImageNet dataset. This forms the foundational layer for further model customisation. Additionally, it allows for the selective freezing of layers (lines 13-14) thus enhancing the

model's adaptability. While the project initially commenced with all layers frozen, the selective freezing was introduced to impact the model's performance. The code then proceeds with a Flatten layer (line 16) to transform the multi-dimensional output from previous layers into a one-dimensional vector. Following this, two fully connected layers are integrated (lines 17-20), and each includes dropout regularisation to combat overfitting. The final component configures the output layer with the appropriate number of neurons based on the dataset's classification requirements (22). In this multi-class classification problem, the output layer is set to contain the necessary number of neurons, accompanied by a "softmax" activation function, enabling straightforward category-based predictions.

3.2. Development process

The process model we adopted in implementing our project is the Cross-Industry Standard Process for Data Mining (CRISP-DM). CRISP-DM is an iterative process that is widely used for managing data mining and machine learning projects. It provides a structured approach while ensuring flexibility and adaptability throughout the project cycle. We implemented this process with overarching agile project management approaches in relation to collaboration and communication, which will be outlined further in '4. Project Management.'

The CRISP-DM model is comprised of six stages that work together in a cyclical manner, as shown in the below in figure 15.

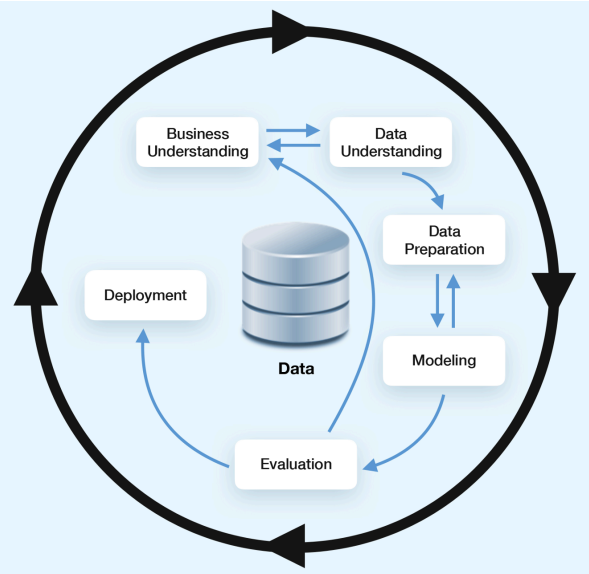

Figure 15. CRISP-DM cycle

Accordingly, our development process was as follows:

- **Business understanding:** The preliminary stage was focused on understanding the problem that we had identified in our Project Requirement and Specifications report. For this project, it required understanding the importance of quick and accurate detection and classification of brain tumours in a clinical setting (see '2.1. Problem summary' for an overview).
- **Data understanding:** This stage involved collecting data, assessing the quality of the data and identifying the relevance of the data to the problem we are looking to solve. Through

research, we were able to identify an appropriate dataset, the Brain Tumour MRI Dataset, that had been validated by similar studies in the area (see '2.2. Project outline' in this report or '2.4. Data availability' in our Project Requirements and Specifications Report for a more detailed outline).

- **Data preparation:** Once the data had been qualified, we had to prepare the data with the goal of the project in mind, which was to classify brain tumours between four classes (meningioma, glioma and pituitary tumours, and no tumours). While we took initial steps such as splitting the data for training, evaluation and testing, we understood that adjustments to the data would need to be made as we progressed through the project.
- **Modelling:** This stage consisted of creating our models, which have been detailed in the previous subsection '3.1. Description of code'.
- **Evaluation:** Once we obtained results, we evaluated the model in accordance with the metrics explored in '2.3. Proposed solutions and evaluation methodologies.' Our results, and comparison between the models, is outlined in '5. Results'. This stage also required us to reflect on whether any of these models were fit for purpose, our overall process, and whether there were steps that could be taken to improve the utility of the project implementation. This is outlined in '6. Discussion and challenges'.
- **Deployment:** Deployment is beyond the scope of our project as it currently stands. As such, this report only provides a review of the project and relevant outcomes.

4. PROJECT MANAGEMENT

4.1. Project management strategies

While our development process was guided by the CRISP-DM process model (as outlined in '3.2. Development process'), it was important that we had general project management strategies in place to ensure the successful completion of the project in the given timeframe. These strategies also facilitated meaningful collaboration and communication between team members. Our approach to working together is outlined below.

Before we started, it was important that everyone on the team understood and agreed on the project details, and that we had a project plan in place. To do so, we had an initial planning meeting to confirm the scope of our project, success criteria, work breakdown structure and internal deadlines. This helped us understand what needed to be done to complete the project on time and to a high standard. It also allowed team members to raise concerns that we could then account for in our project plan. While we finalised these details collectively, we took a flexible approach and noted that we would need to be adaptable to changes that may arise as we continued to work on this project.

To maximise the efficiency of our projects, the tasks were delegated based on individuals' strengths in relation to technical ability. Franciszek, Raeyen, Yang and Yiran worked concurrently on the development of the code for the respective architectures that they were assigned. Sandul and Wendy assisted in interpreting the results, making comparisons to determine model effectiveness, ensuring project outcomes were met, and taking the lead on coordinating the report and presentation. We used a collaborative data notebook called

‘DeepNote’ to ensure everyone had access to the models that were being worked on. This allowed everyone to work to their individual strengths while learning from each other.

The collaboration between the team, and the number of different moving parts of this project, meant that it was essential to maintain clear communication. To do so, we moved from Microsoft Teams to an instant messaging platform. We found that while Microsoft Teams was a convenient tool, it resulted in delays in our communication as team members weren’t used to checking it often. By moving to an instant messaging platform, we were able to reduce response times and collaborate in real-time. Another important aspect of our communication plan was that we made sure to take advantage of the allocated tutorial times to discuss our progress and raise concerns with each other. As people had various commitments throughout the week, planning meetings was difficult but since everyone was available during the tutorial hours, we had to ensure that time was being used effectively.

4.2. Student contribution table

The student contribution table included below outlines the individual contributions of each team member to the project. However, it should be noted that collaboration between the group also required the exchange of ideas, extensive discussions in relation to problem solving, collectively upskilling, and assisting each other with the tasks at hand.

Name	**Outline of contributions**	**Signed**
Franciszek Liszka	● Data preparation, data augmentation, EDA ● Experiments on different architectures and datasets ○ Training of both the initial models CNN1, CNN2, VGG, ResNet, Inception + results ○ Interpretation of the above and model evaluation ● Training of the final models based on the best performing dataset and parameters ● Collaborated to provide explanations for model evaluation and both initial and final results	FL
Raeyen Nuttall	● Development and fine-tuning of the Inception model, explanation & documentation of the code, and the generation of results. ● Collaboration with the team to author sections of the report, specifically focusing on the discussion and challenges related to our research. ● Collaborated to provide explanations and documentation for data preprocessing functions in the project.	Raeyen N
Sandul Mapalagama	● Responsible for finding Project outcomes and challenges. ● Scope for future work. ● Learnings what we gained from this subject.	SHM

Wendy Lam	• Responsible for coordinating the report and presentation, and facilitating collaboration between team members. This included editing the final report, and compiling the video. • Focused on writing the Introduction, Problem Analysis, Project Management and Conclusion • Supported the writing of the CNN code implementation, and overall development process • Supported the analysis and writing of results, including writing the explanation for the model experiments • Supported the analysis and writing of project outcomes • Coordinated with the team to translate the technical information into a readable format	WL
Yang Chang	• Development and fine-tuning of the VGG model, explanation & documentation of the code, and the generation of results. • Collaboration with the team to author sections of the report, specifically focusing on the discussion,challenges and learnings related to our research	YC
Yiran Guo	• Development and fine-tuning of the ResNet model, explanation & documentation of the code, and the generation of results. • Collaboration with the team to author sections of the report, specifically focusing on the discussion,challenges and learnings related to our research	YG

5. RESULTS

5.1. Model experiments

For each model (CNN, VGG, ResNet and Inception), we performed a series of experiments to attempt to increase the accuracy of the classifications. To start, we split the data into three sets: training (60%), evaluation (20%) and testing (20%).

Our first experiment involved exploring augmented data and whether this would yield better results than the original dataset in terms of accuracy. The original dataset had 3780 training samples, 1260 evaluation samples, and 1260 testing samples. For each image, we created 2 additional images using ImageDataGenerator, as seen in Figure 16 below. Examples of augmented data can be seen in Figure 16. As such, our augmented datasets had 11340 training samples, 3780 evaluation samples and 3780 testing samples. We found that using augmented data had a significant impact on the performance of the model as it becomes more

generalisable. We then conducted learning rate testing by testing the performance of the models with different learning rates. Our final experiment was to alter the size of the images (225x225) to determine whether that would produce better results. Specifically, we tried reducing the image size to 115X115 and 80x80 but found that 225 always produced the best results in our models.

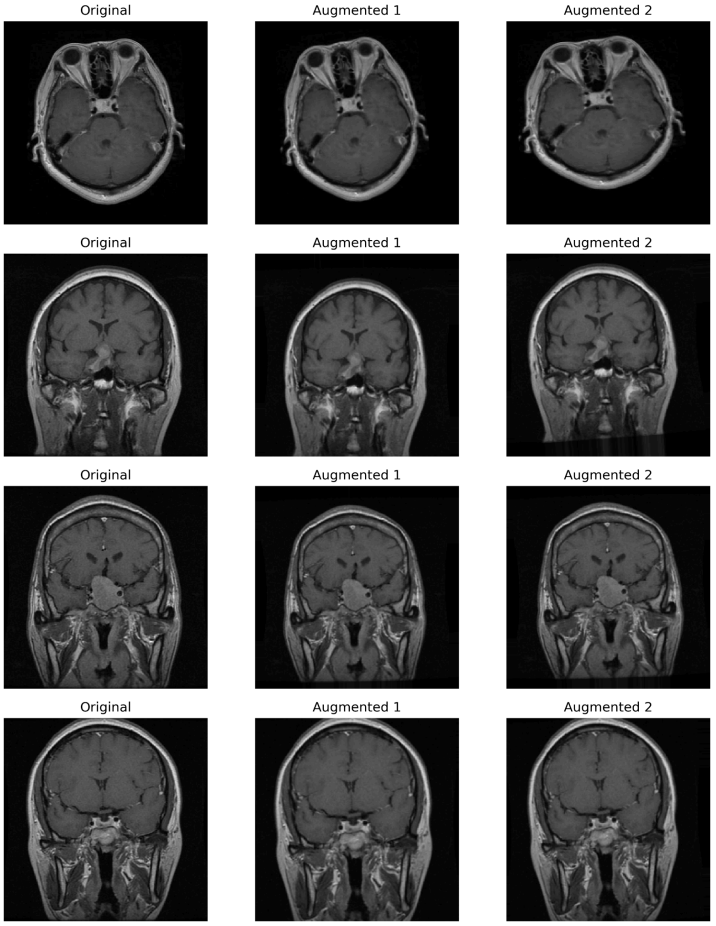

Figure 16. Data augmentation samples

These experiments were conducted using the 60% training set and 20% evaluation set. We then generated learning curves for each model to compare performance. It was determined that the models with a learning rate of 0.0001, image size of 255x255 and used the augmented dataset performed best across each architecture. Once this was established, we merged the training and evaluation data to get a final training set of 80% of the augmented dataset. We trained the model on this data and tested the model on the remaining 20%.

5.2. Model 1: CNN

The progress of the model training can be observed through the loss value over accuracy curve of the model training. Figure 17 shows the performance of both CNN models (CNN 1 and CNN 2) on the original and augmented datasets. As seen in the graph, CNN 2 became more unstable when it was trained on the augmented dataset. This may be due to extreme simplicity of the model, which impacts its ability to generalise well.

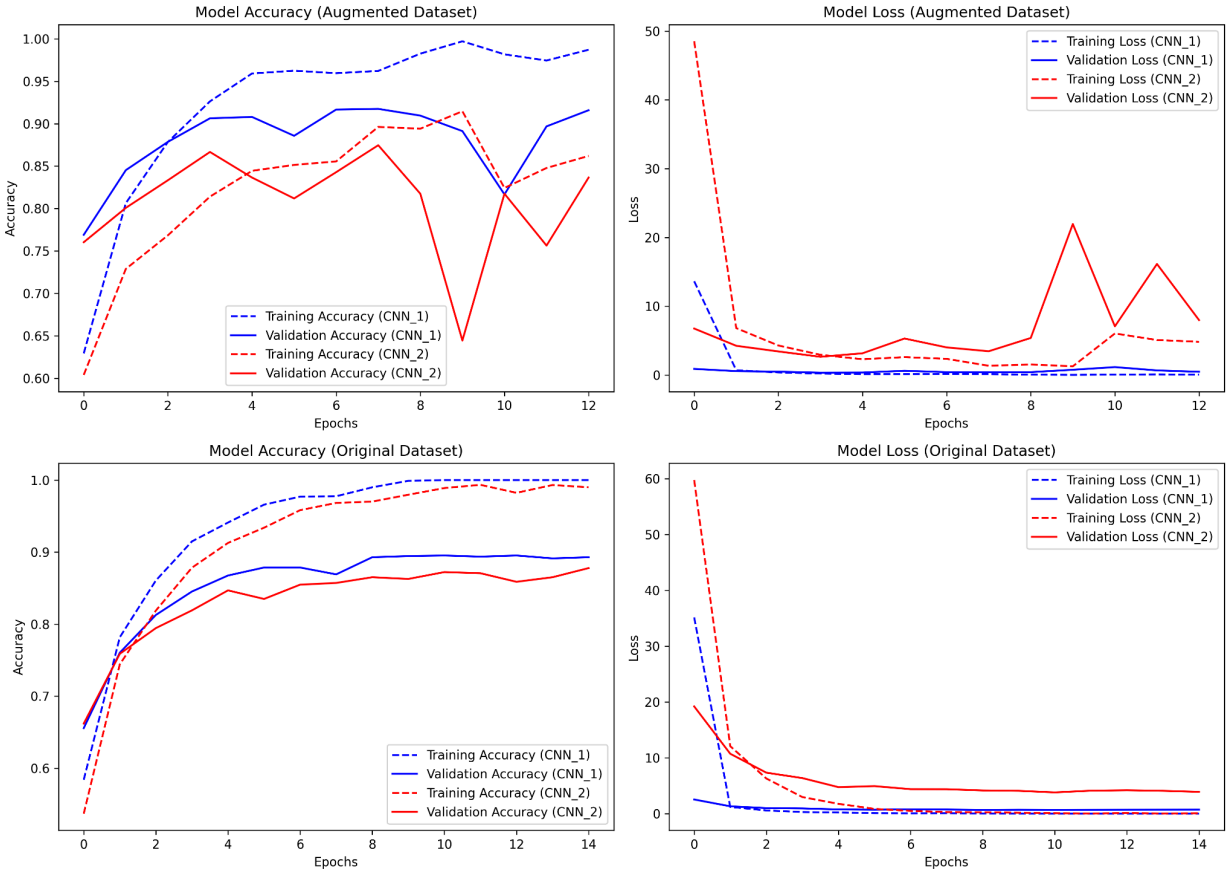

Figure 17. CNN accuracy and loss curve

Accordingly, we focused further experimentation on the CNN 1 model, as seen in Figure 18 below. The model's performance displayed improvements throughout the optimisation process, with the accuracy of the training values reaching a peak at approximately 93%. The convergence of the training and testing curves is promising in terms of optimisation.

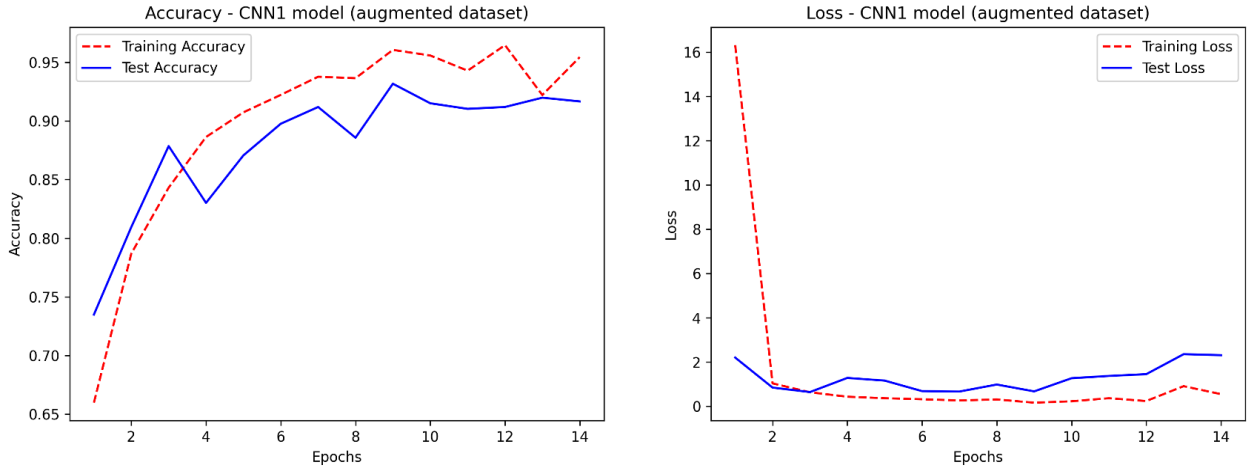

Figure 18. CNN1 accuracy and loss (augmented dataset)

The classification report below (Figure 19) shows the performance of key evaluation metrics, including precision, recall, and F1-score values for each class. The experimental results show

that the model's performance has an accuracy of 93%. The precision metric ranged from 87-99%; the recall metric ranges from 89-96%; and the F1-score ranged from 88-97%. Notably, the numbers indicating lower accuracy were from class 3 (meningioma).

Class	Precision	Recall	F1-Score	Support
Pituitary	0.91	0.96	0.94	233
No Tumour	0.99	0.96	0.97	400
Glioma	0.94	0.92	0.93	324
Meningioma	0.87	0.89	0.88	303
Accuracy			0.93	1260
Macro Average	0.93	0.93	0.93	1260
Weighted Average	0.93	0.93	0.93	1260

Figure 19. CNN 1 classification report

The confusion matrix provides a visual representation of the model's classifications. Observing Figure 20 below, there are considerable misclassifications. For example, within the glioma class, the model has incorrectly predicted samples as meningioma 23 times. On the inverse of that, within the meningioma class, the model has incorrectly predicted samples as glioma 17 times. This suggests the need for a larger dataset for meningioma and glioma, or the need for a tailored approach to address classification accuracy. It's also important to note that the confusion matrix highlights misclassifications in almost all classes.

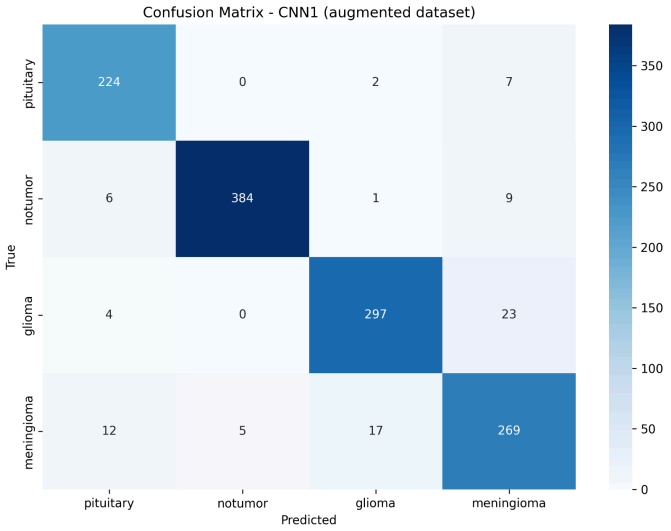

Figure 20. CNN1 confusion matrix

5.3. Model 2: VGG

The progress of the model training can be monitored by looking at the loss value over accuracy curve of the VGG model training. As shown in Figure 21, the performance of the model has improved considerably through the performance test. The graph shows that the accuracy of the training values has been higher than 95%, and the convergence of the training and testing curves is promising in terms of optimisation.

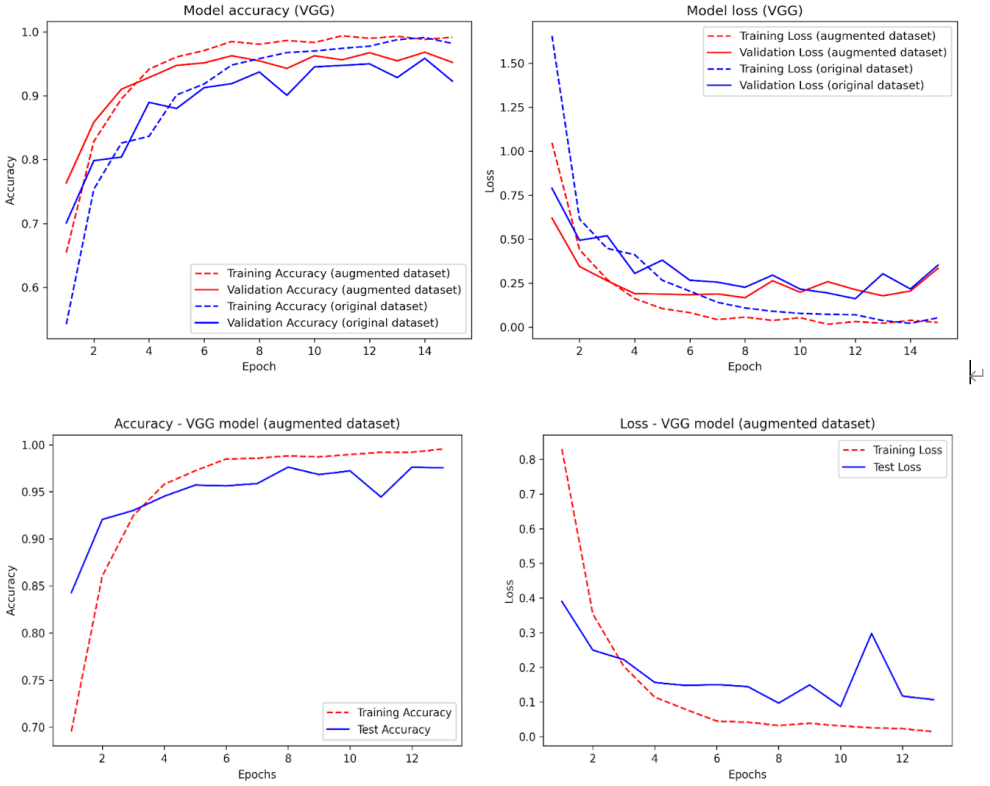

Figure 21. VGG accuracy and loss

The classification report (Figure 22) of VGG shows the performance of key evaluation metrics, including precision, recall, and F1-score values for each class. The experimental results show that the model performs well with an overall accuracy of 98%. For each class, the precision, recall, and F1-score achieved high accuracy rates of above 95%.

Class	Precision	Recall	F1-Score	Support
Glioma	0.99	0.95	0.97	324
Pituitary	0.97	1.00	0.99	233
No Tumour	0.99	1.00	0.99	400
Meningioma	0.95	0.96	0.96	303
Accuracy			0.98	1260
Macro Average	0.97	0.98	0.98	1260

Weighted Average	0.98	0.98	0.98	1260

Figure 22. VGG classification table

The classification results of the model could be quickly viewed by looking at the confusion matrix in Figure 23. This displays the number of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) instances for each class. In the confusion matrix, the number of correctly predicted tumour classifications can be seen in the diagonal blue boxes, and indicates a high number of correct predictions. It is notable that 14 predictions for meningioma were incorrect and classified as glioma.

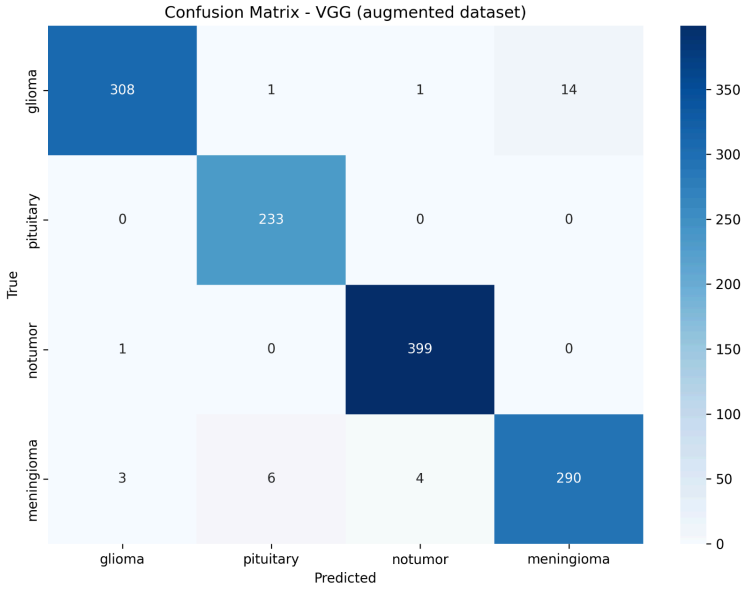

Figure 23. VGG confusion matrix

5.4. Model 3: ResNet

Observing the graphs of loss and accuracy for augmented and raw data in Figure 24, ResNet is very unstable with large fluctuations in loss values and accuracy during the training cycle. This may be due to the fact that ResNet is a more complex model – if the training task is considerably simpler than the application of the model, it may lead to unstable performance. While the model's performance improved using augmented data, it is still relatively unstable before stabilising toward the end of the training cycle.

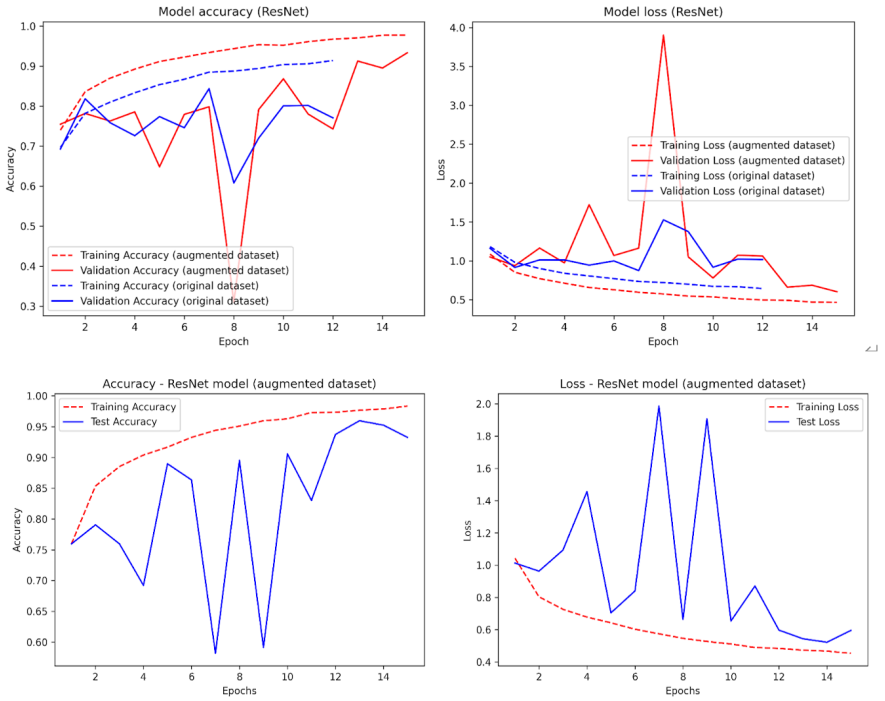

Figure 24. ResNet accuracy and loss

The ResNet classification report (Figure 25) indicates that the model has 93% accuracy, but it is worth noting that meningioma has a precision value of only 79%. This may be related to the presence of more false positive instances.

Class	Precision	Recall	F1-Score	Support
Glioma	1.00	0.88	0.93	324
Pituitary	0.99	0.83	0.91	233
No Tumour	0.99	0.99	0.99	400
Meningioma	0.79	0.99	0.88	303
Accuracy			0.93	1260
Macro Average	0.94	0.92	0.93	1260
Weighted Average	0.94	0.93	0.93	1260

Figure 25. ResNet classification table

The confusion matrix (Figure 26) gives a more visual indication of the number of correct classifications for each category. For example, it can be seen that the model incorrectly predicted 79 samples as having meningioma. This is a high value and explains the lower precision rate for meningioma in the classification report.

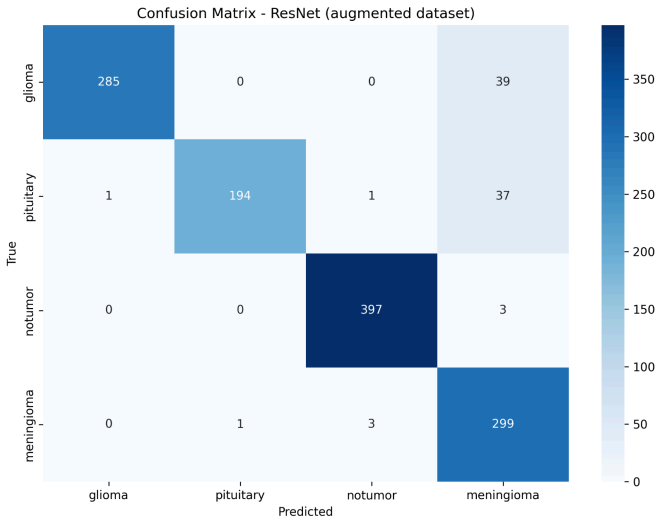

Figure 26. ResNet confusion matrix

5.5. Model 4: Inception

The progress of the model training can be visualised by examining the loss value over accuracy curve of the Inception model training. As illustrated in Figure 27, the model's performance displayed significant improvements throughout the optimisation process. The graph reveals that the accuracy of the training values reached a peak of approximately 98%, showcasing the model's ability to learn and adapt. Additionally, the convergence of the training and testing curves is notably impressive, with both consistently approaching each other. This convergence signifies the successful execution of the model optimisation test and attests to the model's capacity to perform exceptionally well in classifying medical images.

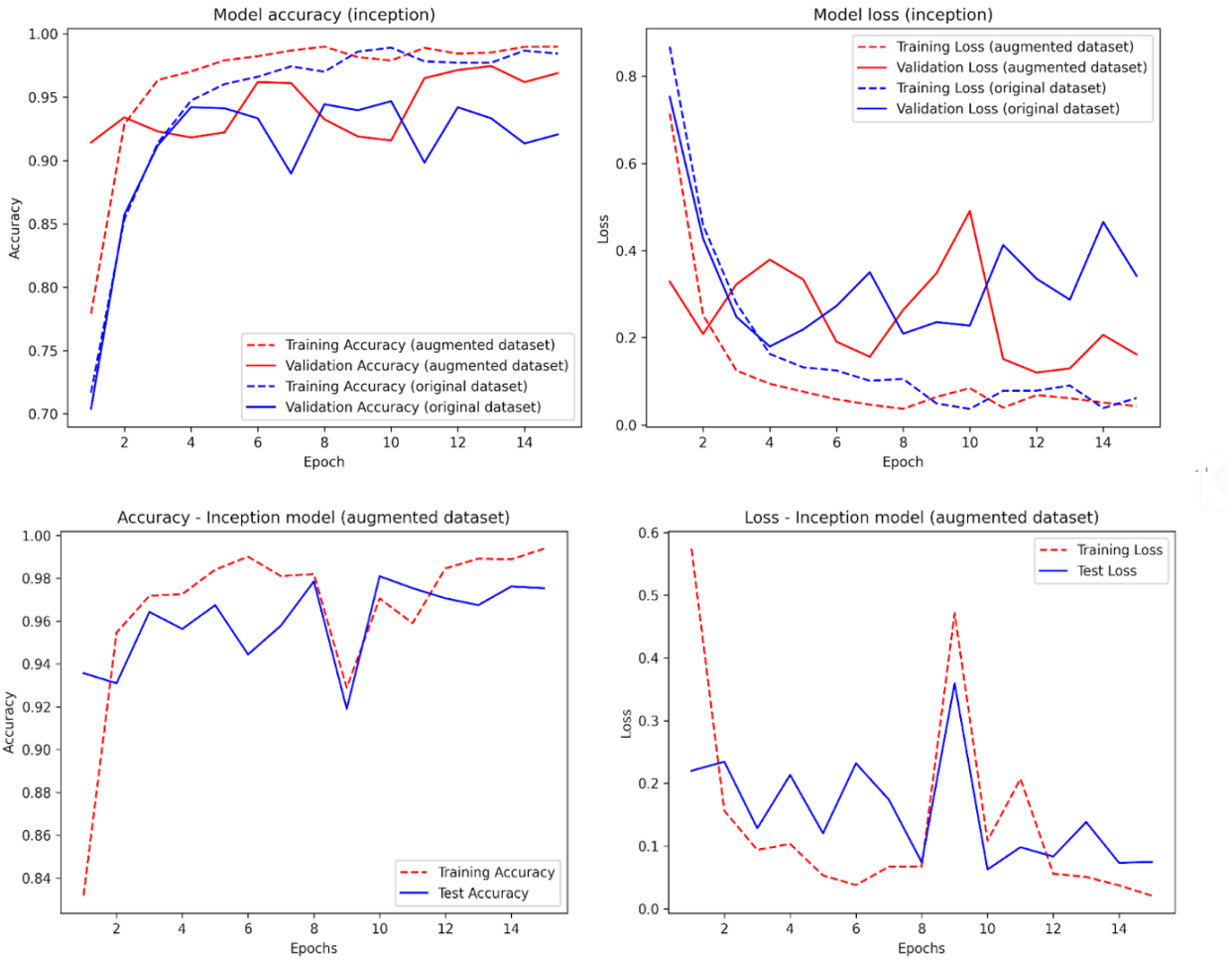

Figure 27. Inception accuracy and loss

The performance report (Figure 28) of the Inception model below, presents an analysis of critical evaluation metrics, including precision, recall, and F1-score values for each class. The experimental findings reveal the model's performance has an accuracy of 98%. The precision, recall, and F1-score metrics for each class consistently reached around 99%, which is notably high. This showcases the model's proficiency in classifying medical images accurately. An interesting observation emerges regarding class 3 (no tumour), where the model's precision is notably lower. This is of particular interest as the original data previously achieved a perfect precision of 1.0 for this class, highlighting the impact of data augmentation on the model's performance.

Class	Precision	Recall	F1-Score	Support
Pituitary	0.99	0.98	0.98	233
No Tumour	0.99	0.99	0.99	400
Glioma	0.99	0.96	0.97	324
Meningioma	0.95	0.98	0.97	303
Accuracy			0.98	1260

Macro Average	0.98	0.98	0.98	1260
Weighted Average	0.98	0.98	0.98	1260

Figure 28. Inception classification table

The confusion matrix below, (Figure 29) provides a visual representation of the model's classifications, highlighting accuracy for each class. In the case of the Inception model, a detailed analysis of the confusion matrix reveals some noteworthy insights. The most prominent misclassification occurs within the "glioma" class, where the model incorrectly predicts instances as "meningioma" 12 times. This observation suggests a specific challenge with the "glioma" class, emphasising the potential need for a larger dataset for this class or a more focused approach to improve classification accuracy. However, it's important to note that the model demonstrates impressive accuracy for the remaining classes, with minimal misclassifications ranging from 3 to 0 instances, underlining its proficiency in correctly identifying various tumour types.

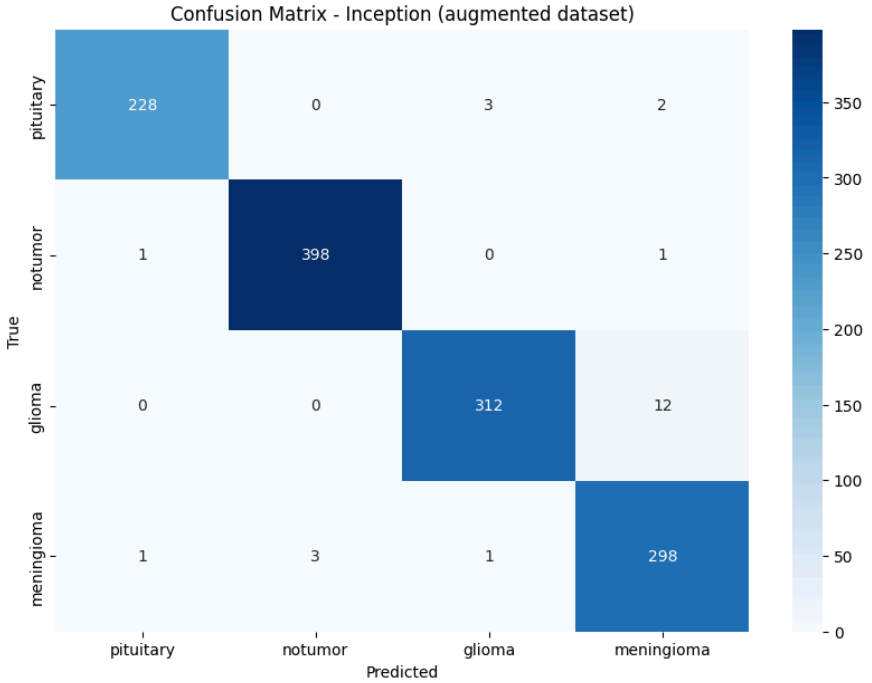

Figure 29. Inception confusion matrix

6. DISCUSSION AND CHALLENGES

6.1. Project outcomes

The aim of this project was to develop a reliable model for detecting and classifying brain tumours from MRI images. To determine the project's success, the model's performance and alignment with standards of accuracy, sensitivity and specificity must be examined. Given the high rates across our evaluation metrics for each model (as outlined in Figure 30 below), we consider the project to be successful in its goal of correctly classifying brain tumours.

Model	Test Accuracy	Precision (Weighted)	Recall (Weighted)	F1-score (Weighted)
CNN (1)	0.932	0.93	0.93	0.93
VGG	0.976	0.98	0.98	0.98
ResNet	0.934	0.94	0.93	0.93
Inception	0.981	0.98	0.98	0.98

Figure 30. Model comparison

As seen in Figure 30, the VGG and Inception models outperformed CNN and ResNet. VGG and Inception both achieved an impressive score of 98% in weighted average metrics across accuracy, precision, recall and F1-score.

With the VGG model, the glioma class had the highest precision of the four classes with 0.99 (and a recall of 0.95 and F1-score of 0.97). The pituitary class showed a perfect recall of 1.00, indicating that there were no false negatives. The training and validation curves for this model showed great convergence, with training accuracy peaking at 97.6%, suggesting effective learning and adaptation. Finally, the confusion matrix showed a relatively low number of misclassifications.

The Inception model showed similarly high precision and recall rates across all classes, reaching 0.99 in almost every evaluation metric. High recall rates are particularly noteworthy in the context of classifying brain tumours as it indicates a lower rate of false negatives. The model had few misclassifications, with the most significant error being 12 instances of glioma misclassified as meningioma. While both models had challenges with the meningioma class, the Inception model achieved a higher recall score for this class. Training and validation curves for this model also displayed great convergence, with training accuracy peaking at 98.1%. This is 0.5% higher than VGG.

In the context of medical imaging for brain tumour diagnosis, the impact of false negatives on patients is particularly high. As such, the selection of the final model should prioritise high recall rates. In this regard, both VGG and Inception exhibit high precision and recall values. However, the Inception model has a slight edge over VGG in certain classes when it comes to recall rate. The Inception model also performed better in minimising the misclassification of images with tumours as no tumour – only three instances of a tumour being present in a scan was misclassified by the Inception model as no tumour, when compared with the 7 misclassifications from VGG.

Given the high stakes involved in brain tumour detection and classification, Inception stands out as the most suitable final selection for an appropriate model. The minor misclassifications made by the Inception model, primarily with glioma being misclassified as meningioma, suggests a need for further fine-tuning. That being said, this does not undermine the overall superior performance of the Inception model in classifying brain tumours. Our recommendation

is to proceed with the Inception model if deployed in a clinical setting, while continuing to collect data to further refine the model.

6.2. Challenges

Throughout the project, our team encountered many challenges that required us to be adaptable and make adjustments to our planned approach. The key issues that affected our project plan were technical ability, computing resource limitations and time constraints. Successfully overcoming these challenges allowed us to complete our brain tumour classification project implementation to a high standard in the given time frame.

In the context of discussing the challenges and insights gained during this project, we initially encountered a significant challenge when we started with all layers of the Inception model frozen. The results were notably unsatisfactory, with an accuracy of only approximately 0.64. This presented a crucial insight into the importance of layer freezing during model training, as we realised that not all layers should remain frozen. The subsequent challenge involved strategically unfreezing the last five layers in the Inception model, aiming to fine-tune its performance. This step contributed to a notable accuracy improvement, reaching around 0.74. However, it was the process of fine-tuning by unfreezing only the first 50 layers, out of a total of 148 in the Inception model, that marked a significant turning point. This decision was pivotal in addressing the challenge of enhancing the model's performance. As a result, our model achieved an impressive final accuracy of 0.96. This progression demonstrates the substantial impact of precise layer freezing and fine-tuning on model performance. It serves as a reminder of the critical role of thoughtful customization of pre-trained models to achieve exceptional results in specific tasks.

Another difficulty we faced was the varying levels of technical knowledge across the team. While some students had prior experience in image processing and pattern recognition, other students were coming across the subject matter for the first time. There were also some people who were more comfortable with Python than MATLAB. To resolve this and allow us to progress with the project, we addressed these issues in our initial meetings and ensured that everyone was transparent about their position in relation to technical ability. From there, we determined that using Python would be best for our team as some people had experience that could be used to guide the team. Each team member also made independent research efforts to better facilitate our understanding of the technical concepts, and we used our time in class to share information and ensure that everyone was on the same page.

As we began the process of experimenting, a difficulty we came across was regarding computer performance and resources, namely the availability of GPUs. This required adjustments to the timeline to ensure that we could complete the project on time. We also had to optimise our computer resources by ensuring our code was efficient, which reduced the time taken to train the models.

Due to the number of experiments and the delays created by a lack of resources, we faced some time constraints. This is especially as there were experiments that took longer than expected to complete. To resolve this issue, we ensure that the experiments were distributed among different team members. We also had to adjust internal deadlines and re-prioritise tasks in our project plan to ensure that we were able to deliver the project on time.

6.3. Scope for future work

In considering the scope for future work in this area, we acknowledge that progress can be made to increase the utility of our implementation for brain tumour classification. One particular area that can be explored is the application of image segmentation techniques to identify the area containing the brain tumour. By isolating the tumour from other brain tissue, models and medical professionals alike can engage in more accurate analysis of a tumour's characteristics. Once the tumour region is determined, other quantitative features of the tumour can be extracted from the segmented region. These features include size, shape, volume and texture. Image segmentation provides valuable information that medical professionals can use in classifying and characterising brain tumours.

6.4. Learnings

Through implementing this project, we were able to deepen our understanding of image processing and pattern recognition techniques. It also provided valuable insight into the development process of building models that are contextualised to a problem, which reflects the work that is done in industry. The project allowed us to explore key technical skills, such as data preprocessing, building, training and evaluating machine learning models, implementing methods to improve model performance, and understanding the design principles of different models to evaluate effectiveness of a model for a given problem.

Data preprocessing was an essential step in implementing our project. It involved cleaning, transforming and organising raw data into a format that was suitable for model training. While we used the 'Brain Tumour MRI Dataset' (as discussed in '2.3. Proposed solutions and evaluation methodologies'), two key preprocessing steps that we found to be important were data augmentation and splitting the dataset. As outlined in '5. Results', data augmentation had a significant impact on the performance of our model as it increased our sample size. Further, splitting the dataset into training, evaluation and test allowed us to better generalise the model.

In regard to building, training and evaluating the machine learning models, a significant amount of learning happened in this stage. A considerable amount of research had to be done to determine potential suitable models. For team members who didn't have much experience with Python, additional research was also required to familiarise themselves with how models could be built. From the initial build, we determined that further experimentation was needed (as outlined in '5.1. Model experiments'). In conducting these experiments, we learned methods that could be applied to improve the performance of a model. Implementing these methods led to optimised results. Finally, understanding the design principles and theoretical application of different models was critical to our ability to evaluate the effectiveness of a model for our brain tumour classification problem. This allowed us to assess whether we achieved our desired project outcomes. It also provided scope to reflect on work that could be done in the future to improve the utility of our models.

7. CONCLUSION

Developments in image processing and pattern recognition can facilitate more efficient and accurate detection of brain tumours within MRI scans, which has traditionally been a highly complex and manual task. By reducing the time and difficulty associated with diagnosing a brain tumour through image processing and pattern recognition techniques, medical professionals are able to make better decisions for their patients in terms of substantiating a diagnosis and providing effective treatment plans sooner rather than later.

Our project focused on the use of convolutional neural networks to achieve accurate identification and classification of brain tumours in MRI scans. In implementing this project, we used different CNN architectures to determine the one most suited to our purpose. Specifically we compared CNNs, VGG, ResNet and Inception. Section 3 provided an overview of our code implementation and development process, which created a strong foundation upon which we could conduct experiments. Section 5 outlined our model experiments and the results we received for each model. Our experiments included data augmentation, testing different learning rates, and changing the sizes of the image. In particular, data augmentation led to considerably improved results across our evaluation metrics. With reference to our models, VGG and Inception performed notably well with a weighted accuracy of 98%. While CNN and ResNet saw improvements when trained on the augmented dataset, there were significant misclassifications. In the context of medical imaging for brain tumours, where the cost of false negatives can be particularly high, Inception had a slight edge in terms of recall which positions it as the preferred model for our problem. That being said, further training and finetuning would increase the accuracy of the Inception model across all classes. To increase the utility of the project, data segmentation techniques could be introduced to locate the area of the brain tumour within the MRI scan.

APPENDIX A: PROJECT CODE

The source code for this project is available on GitHub at the following link:

https://github.com/franekl/brain_tumor_classification.git

APPENDIX B: REFERENCES

Abdusalomov, A. B., Mukhiddinov, M., & Whangbo, T. K. (2023). Brain Tumour Detection Based on Deep Learning Approaches and Magnetic Resonance Imaging. Cancers, 15(16), 4172. https://doi.org/10.3390/cancers15164172

Amin, J., Sharif, M., Haldorai, A., Yasmin, M., & Sundar Nayak, R. (2022). Brain tumor detection and classification using machine learning: a comprehensive survey. Complex Intelligence Systems, 8, 3161. https://doi-org.ezproxy.lib.uts.edu.au/10.1007/s40747-021-00563-y

Amin, J., Sharif, M., Raza, M., Saba, T. & Anjum, M. (2019). Brain tumor detection using statistical and machine learning method. Computer Methods and Programs in Biomedicine, 177, 69. https://doi-org.ezproxy.lib.uts.edu.au/10.1016/j.cmpb.2019.05.015

Nickparvar, M. (2021). Brain Tumour MRI Dataset. Kaggle. 10.34740/kaggle/dsv/2645886

Provost, F., & Fawcett, T. (2013). Data Science for Business: What You Need to Know About Data Mining and Data-Analytic Thinking. O'Reilly Media.

Sapra, P., Singh, R., & Khurana, S. (2013). Brain Tumour Detection Using Neural Network. *International Journal of Science and Modern Engineering*, 1(9), 2319. https://www.ijisme.org/wp-content/uploads/papers/v1i9/I0425081913.pdf

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427-437.