



SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

60.001 Applied Deep Learning 2024

Brain Tumour Classification

Ayu Permata Halim Mendoza	1006069
Rio Chan Yu Hoe	1005975
Parigya Arya	1006096

Executive Summary

This summary provides an overview of a brain tumour classification project, covering various stages from problem scoping to model optimization techniques.

This project utilises a classification dataset from Kaggle. The project begins with scoping the problem and collecting and preprocessing the dataset. It outlines the required inputs and outputs, followed by the selection and preprocessing of the dataset, including data transformation, loading, and visualization. The base model selection and training phase explore the implementation of a naive CNN, along with considerations such as batch size, epoch selection, and model architecture. Further experimentation involves integrating elements from advanced architectures like DenseNet121, VGG, Xception, and ResNet into the base model to improve performance. Additionally, optimization strategies such as optimiser selection, hyperparameter tuning, data augmentation, and regularization methods are employed to enhance the model's accuracy and generalization. The project concludes with suggestions for further improvements and a comprehensive discussion in the conclusion section. Overall, the project aims to develop an efficient brain tumour classification model through rigorous experimentation and optimization techniques.

Table of Contents

Introduction and Problem Scoping.....	5
Dataset Collection and Preprocessing.....	7
Required Inputs and Outputs.....	7
Dataset Selection	7
Dataset Preprocessing	7
Data Transformation.....	7
Data Loading	7
Data Visualisation	7
Base Model Selection and Training.....	9
Naive CNN.....	9
Batch Size and Epoch Selection.....	9
Layers Explanation.....	9
Architecture Experimentation	12
DenseNet121	12
VGG	13
Xception	14
ResNet.....	15
Improving our Base Model	17
Implementation of Elements from Xception and ResNet into Base Model	17
Optimiser Selection	18
Hyperparameter Tuning on Naïve CNN Model	20
Data Augmentation on Naïve CNN Model	23
Regularisation Methods on Naïve CNN model.....	24
Further Improvements	26
Implementation of a more complex dataset to fine tune the naive CNN model	26
Implement deep layers on the CNN Model.....	26
Experimenting on more combinations of hyperparameter tuning using GridSearch	26
Experimenting on a better combination of data augmentation.....	26
Employing early stopping for epoch: To find the optimal balance between model complexity and generalization and prevent overfitting.....	26
Employing different regularization methods to generalise the model.....	26
Using Initializers (He, Xavier) and Batch Normalisation.	26
Conclusion	27

References.....	28
Annexes	30
Annex 1	30

Introduction and Problem Scoping

The brain, recognized as the central organ of the human nervous system, plays a pivotal role in human physiology. In the context of a dynamically changing global landscape, escalating environmental pollution, and the ongoing evolution of human biology, there has been a documented rise in the incidence of brain cancer. Notably, global statistics reveal a significant increase in brain cancer rates for both males (AAPC = 0.4%, 95% CI = 0.4–0.5) and females (AAPC = 0.5%, 95% CI = 0.5–0.6) (Ilic & Ilic, 2023). Brain tumours are an abnormal growth of cells in the brain, also known as multifactorial groups of neoplasm, and if they proliferate, they can lead to the development of brain cancer (Rasheed, Rehman, & Akash, 2021).

Therefore, early and accurate detection of brain tumours is crucial for initiating timely interventions and improving patient outcomes in the face of this rising global health challenge. It is imperative to continually enhance existing medical imaging devices and models utilised in the detection of brain tumours, emphasising the critical need for early identification. Moreover, research indicates that the detection and diagnosis of cancers or tumours exhibit an estimated median error rate of 4.4%, with variations ranging from 0.9% to 18.2%, contingent upon specific cancer types (Newman-Toker et al., 2020). This underscores the importance of advancing and refining models employed in cancer detection to enhance the precision and accuracy of early detection. This is particularly crucial in the context of brain tumours because they can manifest with diverse and subtle characteristics, making them challenging to identify accurately. Additionally, early detection allows for prompt initiation of treatment, which can significantly impact patient prognosis and their quality of life.

In response to these considerations, the development and ongoing refinement of a deep learning model designed for the classification and identification of brain tumours utilising medical imaging devices, such as MRI, is an important step in the advancement of the medical field. The primary objective of this model is to accurately differentiate various types of brain tumours.

There are many different types of brain tumours, and for this project, we will be focusing on categorising into the three most common types of brain tumours, which are — Glioma, Meningioma, Pituitary Gland Tumour, as well as No Tumour. Gliomas represent 24% of all central nervous system (CNS) tumours and are therefore the most common primary brain tumours of the CNS (Delobel et al., 2023). They can be highly aggressive, hence requiring prompt and targeted intervention (Zhao et al., 2023). Meningiomas, on the other hand, develop from the meninges and tend to be slow growing, but can still pose serious health risks depending on their location and size (Jeffrey et al., 2024). Pituitary gland tumours arise from the pituitary gland and can affect hormone regulation, necessitating specialised treatment approaches (SingHealth, 2024).

Therefore, categorising these tumours accurately is essential as it enables clinicians to tailor treatment strategies to each patient's specific condition, maximising the likelihood of successful outcomes. Additionally, distinguishing between different types of brain tumours can also aid in prognosis determination and monitoring treatment response over time, ultimately leading to more personalised and effective patient care.

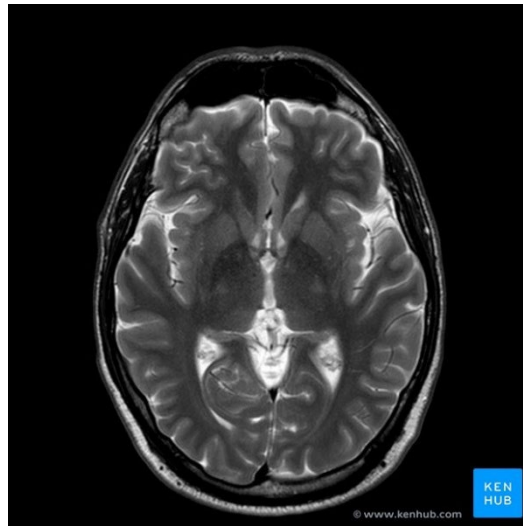


Fig 1. Image of a Healthy Brain
(Vaskovic, 2023)

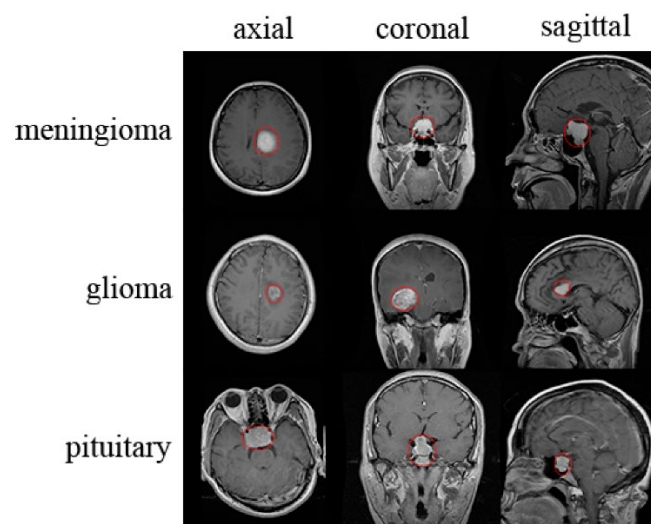


Fig 2. Images of Brains Containing Tumours

Dataset Collection and Preprocessing

Required Inputs and Outputs

The inputs to our model would be images of brain tumours that have been detected through MRI scans.

Henceforth, based on each tumour's visual characters and features extracted during training, our output would be the predicted class or type label of the brain tumour — Glioma, Meningioma, Pituitary Gland Tumour and No Tumour (meaning healthy brain).

Dataset Selection

To train our model, we will be using a labelled dataset from Kaggle - [Brain Tumor MRI Dataset](#), which is a combination of the following datasets: [figshare](#), [SARTAJ dataset](#) and [Br35H](#). We decided to choose this as our training dataset due to several factors. Firstly, it contains a substantial number of images, totalling 7023, which provides a diverse and extensive set of examples for training our model. Additionally, this dataset is well-documented, offering comprehensive information about the image sources and classifying them into 4 classes — glioma, meningioma, no tumour and pituitary. This level of documentation is crucial for ensuring transparency and reproducibility in our research.

Compared to other available datasets, such as the [Brain Tumor Image Dataset](#), which consists of 3064 T1-weighted contrast-enhanced images from 233 patients, the [Brain Tumor MRI Dataset](#) offers a larger and more varied collection of images. Furthermore, while other datasets may focus on specific types of tumours or lack balance across classes, the [Brain Tumor MRI Dataset](#) includes a more balanced distribution across its four classes: glioma, meningioma, no tumour, and pituitary tumour. This balance ensures that our model is trained on a representative sample of each class, enhancing its ability to accurately classify tumours in unseen data. Therefore, considering the importance of data quality and diversity in training robust machine learning models, the [Brain Tumor Image Dataset](#) emerged as the most suitable choice for our project. Its combination of ample documentation, large number of images, and balanced class distribution provides a solid foundation for training our CNN model to classify brain tumours with high accuracy and reliability.

Dataset Preprocessing

This section contains a mix of data transformation, data loading and data visualisation. All these steps are necessary to prepare a good dataset to be used to feed into our model to attain a good accuracy and prevent overfitting by understanding demographics of our data and removing biasness.

Data Transformation

Data is Initially Downloaded from Kaggle, in which there is a need to transform these images as there exist inconsistencies with their image shapes. This is an important step as the model would assume inputs of the same size to pass through it. Therefore, we have to transform images to the same size to fit the particular model. We have set images to 299 and 224, most of our models will run on 299,299 while vgg16 would run on 224 resolutions. This will be further explained in the further sections of our report.

Data Loading

The dataset has already been split into train and test; therefore, we loaded these split datasets with Pytorch data loader datasets into batch sizes. We varied the batch size to understand the relationship with the model and will be explained in detail.

Data Visualisation

Having transformed and loaded the data, we need to visualise these aspects to see that the data we manipulated are useful for our model. Firstly, we look at the images to see if they are properly cropped to the right image

sizes and the images are not corrupted. Then we also look at the demographics of data split by the data loader to ensure that dataset is split well for training and evaluating. This makes our dataset less biased, usable, and ready to be fed into the models.

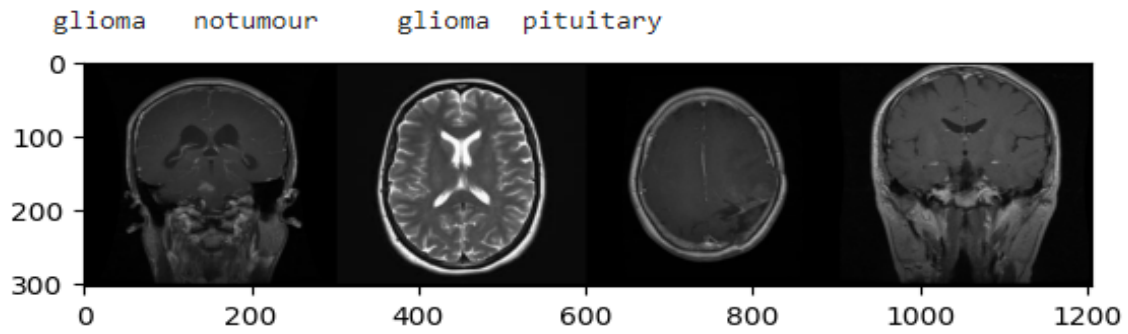


Fig 3. Image Visualisation

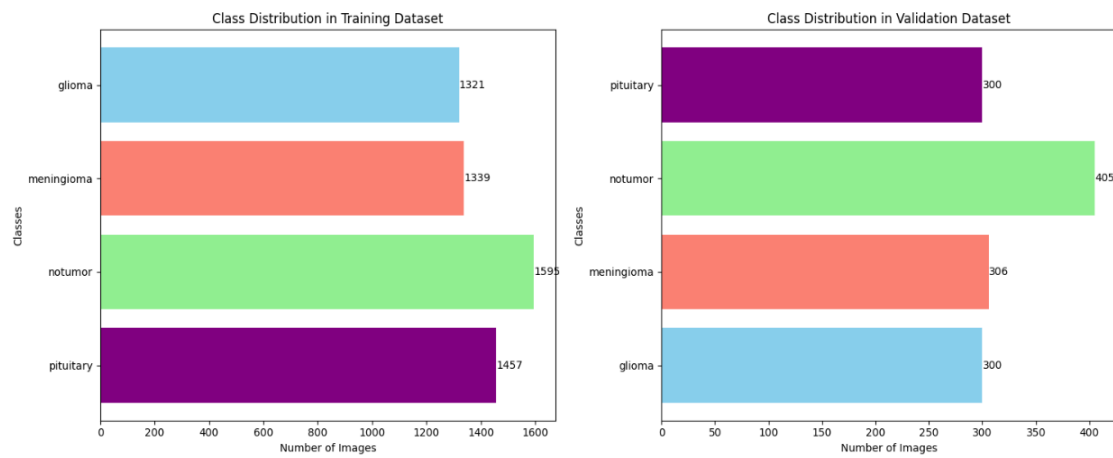


Fig 4. Class Distribution of Training Datasets

Base Model Selection and Training

For our model, we will be focusing on Convolutional Neural Networks (CNNs), which is the most effective approach for brain tumour detection and categorisation due to their specialised architecture and inherent capabilities in processing medical imaging data. CNNs are specifically designed for image processing and recognition tasks, allowing them to automatically learn hierarchical feature representations from raw input images with minimal preprocessing. Inspired by the connectivity pattern in the human visual cortex, CNNs excel at capturing spatial relationships and patterns in images, making them particularly well-suited for analysing complex medical images such as brain MRI scans (Datagen, n.d.).

Naive CNN

Convolutional Neural Networks (CNNs) comprises of 5-layer architecture to excel in image processing. Convolutional layers with filters extract features like edges. Pooling layers reduce complexity. Fully connected layers then performs classifications based on the learned features. Dropout layers help prevent overfitting, while activation functions like ReLU introduce non-linearity for complex pattern recognition. (Shah et al., 2017)

Convolutional Neural Networks (CNNs) rely on a delicate balance between depth and width for optimal performance. While deeper networks, with more convolutional and fully connected layers, can learn intricate relationships and achieve higher accuracy (Bengio, 2013; Khan et al., 2020), they are susceptible to overfitting and vanishing gradients (He et al., 2016). Conversely, wider networks, characterized by a larger number of channels per layer, extract richer features and are necessary for theoretical properties like universal approximation (Hanin & Sellke, 2017). However, excessive width increases computational cost and can even degrade accuracy (Qiu et al., 2020). Finding the optimal balance between depth and width is crucial for building effective CNN models. (Chen & Tsou, 2022)

Before testing other models, we decided to build a naive CNN model from scratch so that we could better understand the fundamentals of CNN, as well as our chosen the dataset. It also allows the team to research and understand the model architecture better to help with model customisation and adaptation. It will also be easier to appreciate other pre-trained models and understand which pre-trained models would be best suited for the dataset.

Batch Size and Epoch Selection

After building the naive CNN model, we experimented with different batch sizes to determine which batch size is most suited for this project. We experimented manually using batch sizes of 16, 32 and 64 and compared their results. A batch size higher than 64 was not needed because even though it would allow for faster convergence time, our dataset was relatively small with only 7000+ images thus it does not require that high of a computational time.

After experimenting with the batch sizes, batch size 16 was chosen because it caused the model to overfit the least using the Naive CNN. Even though the computational time is the slower, since the dataset is relatively small, the slow training time can be considered negligible.

For the Epoch size, Epoch size 20 was selected as it is easier to run a comparison with the original on Kaggle. Although, the model does overfit, further improvements can be made to prevent this overfitting such as using regularisation methods or early stopping.

Layers Explanation

The naive CNN was built using the following code:

```

model = nn.Sequential(
    nn.Conv2d(3, 32, 3),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 32, 3),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, 3),
    nn.ReLU(),
    nn.MaxPool2d(2, 2)
)

output_size = calsize(model, channum, img_height, img_width)

layers = list(model.children())
layers.extend([
    nn.Flatten(),
    nn.Linear(output_size[0] * output_size[1] * output_size[2], 64),
    nn.Dropout(0.5),
    nn.Linear(64, 4),
    nn.Softmax(dim=1)
])

model = nn.Sequential(*layers).to(device)

```

Fig 5. Naive CNN Model Code

1. **Convolutional Layer 1:** It starts with a convolutional layer (**nn.Conv2d**) with an input channel size of 3 (which corresponds to RGB images), output channel size of 32, and a kernel size of 3x3. This layer extracts 32 feature maps from the input image.
2. **ReLU Activation:** Following each convolutional layer, a rectified linear unit (ReLU) activation function (**nn.ReLU**) is applied element-wise to introduce non-linearity to the model.
3. **Max Pooling Layer 1:** After the activation, a max-pooling layer (**nn.MaxPool2d**) with a kernel size of 2x2 and a stride of 2 is used to down sample the feature maps by selecting the maximum value from each 2x2 region. This reduces the spatial dimensions of the feature maps by half.
4. **Convolutional Layer 2:** Another convolutional layer with 32 input channels (matching the output channels of the previous layer), 32 output channels, and a kernel size of 3x3 is applied.
5. **ReLU Activation:** The ReLU activation function is applied again.
6. **Max Pooling Layer 2:** Another max-pooling layer is applied, similar to the first one, further reducing the spatial dimensions.
7. **Convolutional Layer 3:** This time, a convolutional layer with 32 input channels and 64 output channels is applied.
8. **ReLU Activation:** The ReLU activation function follows.
9. **Max Pooling Layer 3:** Another max-pooling layer is applied.

ReLU activation function was selected as it introduces non-linearity to the model, which allows it to learn complex patterns and relationships within the data.

When training has been completed, the following graph was obtained with ***final training accuracy: 0.9142156862745098 and final validation accuracy: 0.8489702517162472***. This final value can be further improved in other iterations of this project.

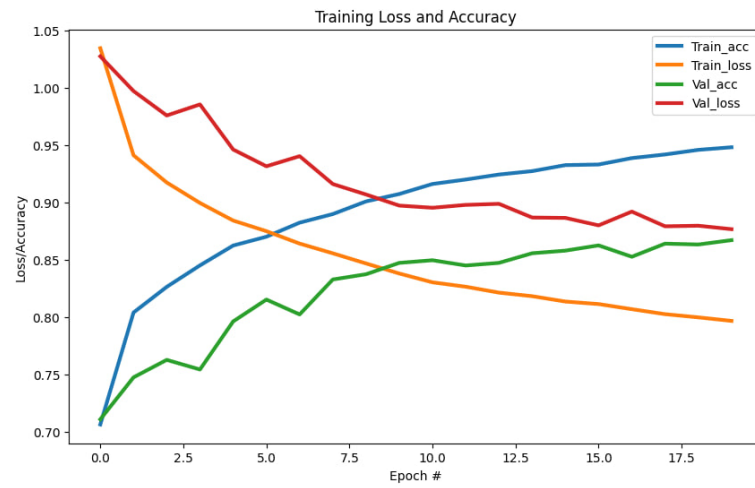


Fig 6. Loss and Accuracy Function for Naïve CNN

Architecture Experimentation

We look towards exploring state-of-the-art models that are used in current applications of image recognition and more specifically Image recognition in the field of healthcare. We apply transfer learning to understand how some of these models perform for our dataset. Models were trained on similar hyperparameters for the base models and fine-tuned when necessary.

Table 3

Evaluation performances of deep learning models and the proposed model.

Models	PR (%)	RE (%)	SE (%)	SP (%)	AC (%)	F1-Score (%)
Xception	95.7	95.9	95.9	95.4	95.6	95.8
InceptionResNetV2	96.2	96.6	96.6	96.1	96.3	96.4
ResNet50	96.6	96.8	96.8	96.2	96.5	96.7
InceptionV3	96.7	97.1	97.1	96.3	96.4	96.9
VGG16	97.4	97.7	97.7	97.3	97.6	97.5
EfficientNet	97.7	97.9	98.0	97.5	97.8	97.8
The proposed model	99.5	99.3	99.3	99.4	99.5	99.4

Fig 7. Evaluation Performance Table

(Abdusalomov et al., 2023)

DenseNet121

The DenseNet structure, proposed by Huang et al., consists of dense blocks where each layer is connected to all preceding layers. This connectivity pattern allows for efficient information flow and feature reuse, addressing vanishing gradient issues and reducing the number of parameters. DenseNet-121, a popular variant, comprises multiple dense blocks separated by transition layers that down-sample feature maps. It employs batch normalization, ReLU activation, and convolution operations within each layer. DenseNet's implication includes its need for a large training dataset to achieve optimal performance due to its depth and parameter efficiency. To mitigate this requirement, transfer learning (TL) is often employed, where models pretrained on large datasets like ImageNet are fine-tuned on target tasks. TL not only enhances performance but also reduces overfitting and training time while requiring fewer labelled samples. Thus, DenseNet's structure and implications underscore its effectiveness in various domains, particularly when coupled with TL techniques.

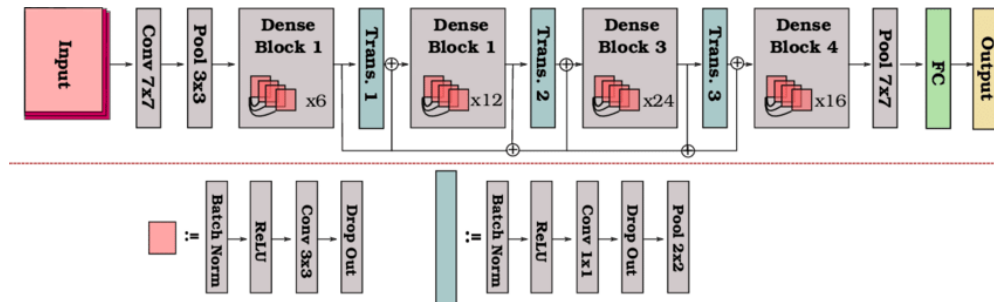


Fig 8. DenseNet 121 Architecture

(Radwan, 2019)

Loss: 0.8875, Validation Loss: 0.8967, Validation Accuracy: 84.90%, Validation F1 Score: 0.8463, Validation Precision: 0.8473, Validation Recall: 0.8490

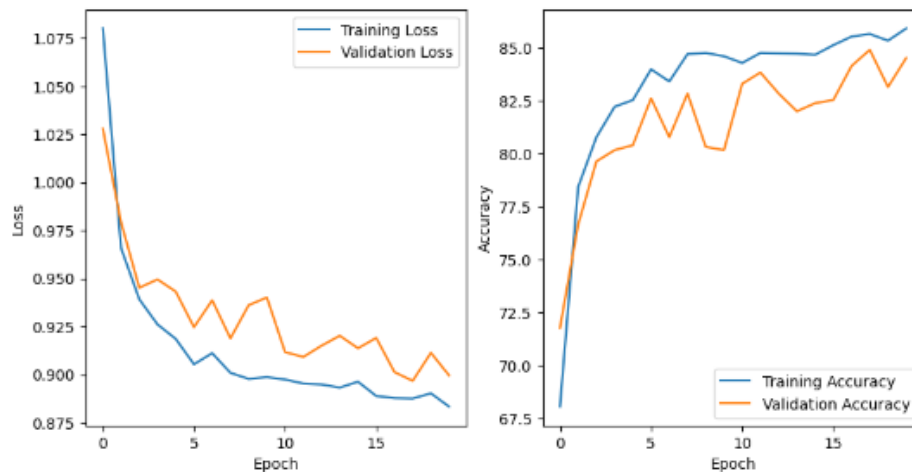


Fig 9. Image of DenseNet121 Loss and Accuracy Function

VGG

VGG16 is a deep convolutional neural network renowned for image classification. With 16 layers, it employs 3x3 convolutional filters and 2x2 max-pooling layers consistently. Despite its effectiveness, its large size—138 million parameters—poses challenges in terms of computational resources and memory requirements. Additionally, VGG16 typically operates on images resized to 224x224 pixels, establishing a standard input size that facilitates compatibility and efficiency in processing.

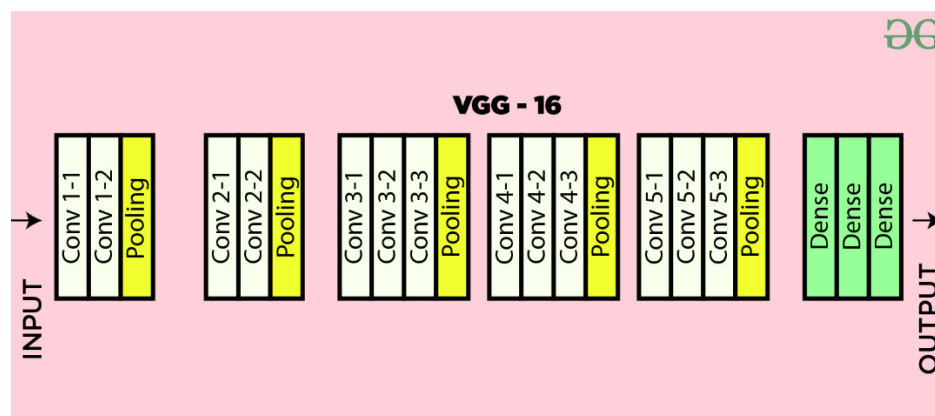


Fig 10. VGG16 Architecture

(GeeksforGeeks, 2024)

```

img_height, img_width = 224, 224 |

train_data_dir = '/content/Training'
validation_data_dir = '/content/Testing'

classes = ('glioma', 'meningioma', 'notumour', 'pituitary')

transform = transforms.Compose([
    transforms.Resize((img_height, img_width)),
    transforms.CenterCrop((img_height, img_width)),
    transforms.ToTensor()
])

```

Fig 11. Code Snippet of Changed Image Size

Loss: 0.9492, Validation Loss: 0.9363, Validation Accuracy: 80.55%, Validation F1 Score: 0.8019, Validation Precision: 0.8055, Validation Recall: 0.8055

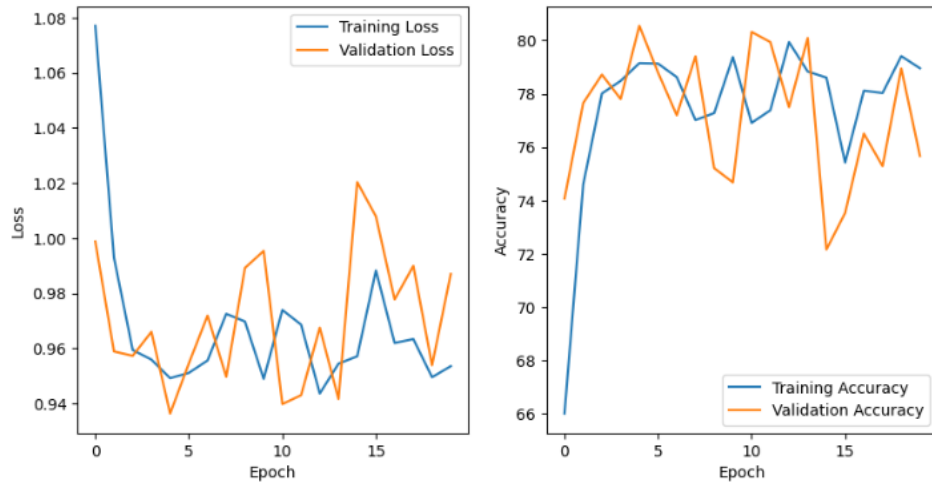


Fig 12. Image of VGG16 Loss and Accuracy Function

Xception

Xception leverages depth-wise separable convolutions, a technique that breaks down standard convolutions for improved efficiency. This modular network, built with repeated processing blocks, flows through entry, middle, and exit stages. While it boasts lower computational cost compared to traditional architectures, Xception achieves competitive accuracy on image classification tasks (Chollet, 1970). This makes it a compelling choice for resource-constrained environments or mobile deployment, though the trade-off is potentially lower peak accuracy compared to more complex models.

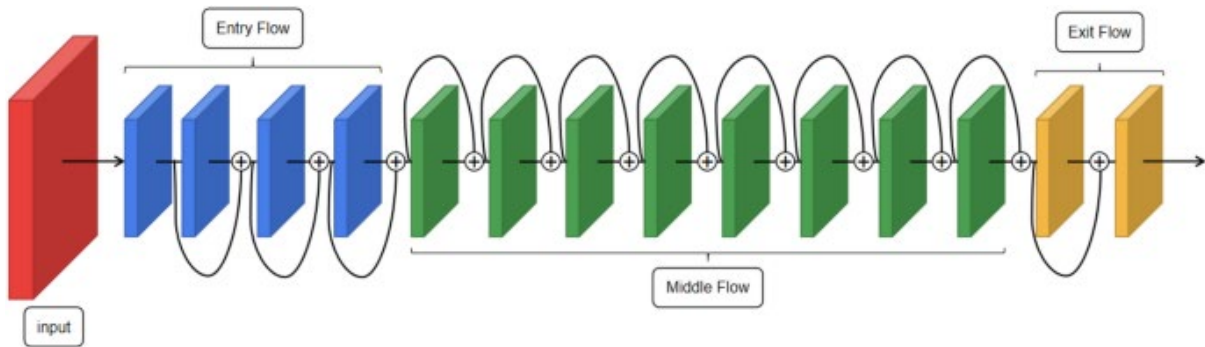


Fig 13. Xception Architecture

(Liu et al., 2022)

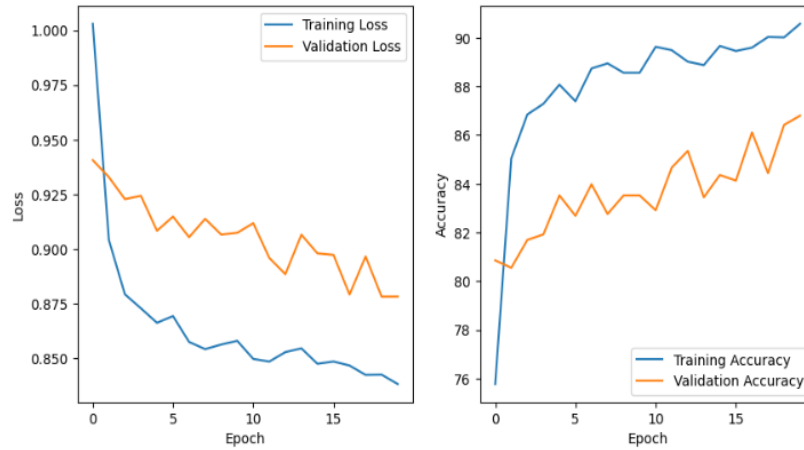


Fig 14. Image of Xception Loss and Accuracy Function

Loss: 0.8382, Validation Loss: 0.8782, Validation Accuracy: 86.80%, Validation F1 Score: 0.8644, Validation Precision: 0.8697, Validation Recall: 0.8680

ResNet

To experiment with Residual Neural Network (ResNet), we leveraged transfer learning, employing pre-trained ResNet models, specifically ResNet18 and ResNet50, as starting points due to their proven effectiveness in various computer vision tasks.

The training process involved a comprehensive hyperparameter search using grid search methodology to determine the optimal combination of learning rate (LR) and weight decay (WD) (See Figure 15). We experimented with a range of LR values including 0.01, 0.001, and 0.0001, as well as WD values of 0.01, 0.001, and 0.0001. Additionally, we evaluated both ResNet18 and ResNet50 architectures to discern which model configuration yielded the highest validation accuracy.

```
# Hyperparameters and models to try
learning_rates = [0.01, 0.001, 0.0001]
weight_decays = [0.01, 0.001, 0.0001]
models_to_try = ['resnet18', 'resnet50']

# Grid search
best_val_acc = 0.0
best_hyperparams = None
best_model = None
```

Fig 15. Resnet Hyperparameters

To facilitate model training and evaluation, a custom training loop was utilised that encompassed both training and validation phases (See Figure 16). During training, the Adam optimizer was employed with the selected LR and WD values, while monitoring the model's performance using the Cross Entropy Loss function. To prevent overfitting and enhance generalization, we incorporated a *ReduceLROnPlateau* scheduler, dynamically adjusting the learning rate based on validation loss trends.

Following each training run, we visualized the training results using line plots to gain insights into the model's learning dynamics and performance trends over time. These visualizations provided valuable insights into the effectiveness of different hyperparameter configurations and model architectures (Refer to the ipynb document '*ResNet Hyperparameter Tuning*').

Ultimately, we managed to identify the best-performing model configuration, characterized by the highest ***validation accuracy of 0.995***.

```
Best hyperparameters: {'lr': 0.0001, 'wd': 0.001, 'model': 'resnet18'}  
Best validation accuracy: 0.9946605644546148
```

Fig 16. Best Hyperparameters Obtained

Improving our Base Model

After experimenting with various architectures in the previous section, we then embarked on enhancing our base model to address its low accuracy as compared to models found online. To achieve this, we sought inspiration from successful transfer learning approaches, particularly from architectures like Xception and ResNet as these 2 models achieved the highest validation accuracy during our transfer learning experiment. We incorporated elements from these models into our naive CNN architecture. Additionally, we explored techniques such as incorporating an optimiser, hyperparameter tuning and data augmentation to further improve our model's performance.

Implementation of Elements from Xception and ResNet into Base Model

We felt that Resnet and Xception were better suited for our task and tried to implement them together. By understanding the intricate architecture of both models, we assimilated them adding onto significant characteristics from each feature. From Resnet, we adopted the residual blocks and from Xception we added the flow sequences.

```
self.entry_flow = nn.Sequential(
    nn.Conv2d(in_channels, 64, kernel_size=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True)
)

# Middle Flow (with residual blocks)
self.middle_flow = nn.Sequential(
    nn.Conv2d(32, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.Conv2d(32, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.Conv2d(32, 32, kernel_size=3, padding=1), # Residual block 1
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.Conv2d(32, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.Conv2d(32, 64, kernel_size=3, padding=1), # Residual block 2
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(64, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2)
)

# Exit Flow (inspired by Xception)
self.exit_flow = nn.Sequential(
    nn.AvgPool2d(2, 2)
)

# Fully connected layers
self.fc = nn.Sequential(
    nn.Flatten(),
    nn.Linear(in_features=87616, out_features=64),
    nn.Dropout(0.5),
    nn.Linear(64, num_classes),
    nn.Softmax(dim=1)
)
```

Fig 17. Code Snippet of Xception and Resnet Model

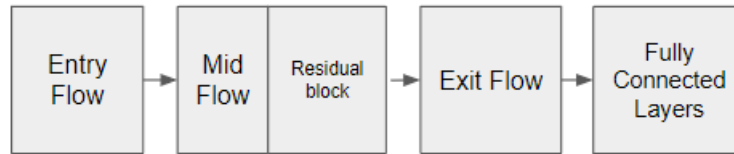


Fig 18. Visual representation of Architecture

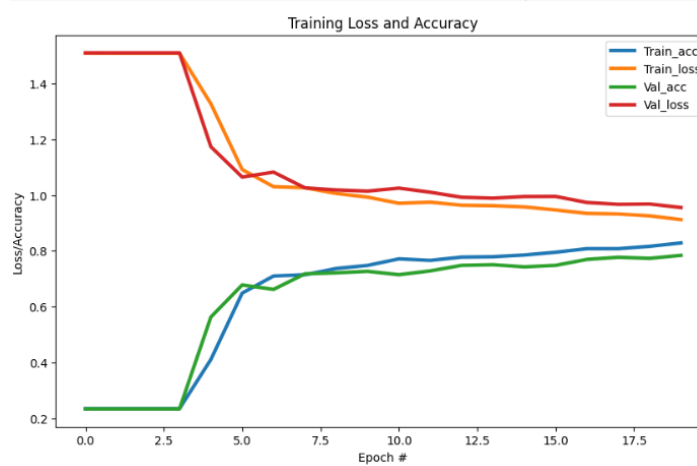


Fig 19. Image of Hybrid Model Loss and Accuracy Function

Final training accuracy: 0.8287815126050421 Final validation accuracy: 0.7841342486651411

Upon scrutinizing the performance, it became evident that our model fell short of expectations when compared to our baseline. This discrepancy could stem from various factors, such as the model's excessive depth for a relatively straightforward dataset, potentially leading to insufficiently trained features. Additionally, suboptimal learning rates might have hindered convergence, resulting in sluggish and ineffective training processes.

Nevertheless, there's a silver lining to this setback: our model exhibits a promising learning curve, as indicated by the close proximity of train and validation accuracies, suggesting minimal overfitting. However, it's crucial to acknowledge the significant computational expense and time investment associated with training this model. In contrast, our simpler baseline model achieved higher accuracy in a fraction of the time.

Consequently, we've made the strategic decision to pivot towards fine-tuning our baseline model rather than persisting with the transfer learning approach. Nonetheless, this experience has provided valuable insights into the amalgamation of state-of-the-art model characteristics, paving the way for potential innovations in the future.

Optimiser Selection

In our investigation of various optimizers with default learning rates, we sought to determine the most suitable optimizer for our specific convolutional neural network (CNN) task. Despite lacking visual aids, our findings provide compelling evidence to support our optimizer selection.

The optimizers and their respective parameters utilized in our experimentation were as follows:

Optimiser	Learning Rate	Average Validation Loss	Average Validation Accuracy
Adam	0.01	0.9066	0.8368

SGD	0.01	1.0207	0.7285
RMSprop	0.01	1.0207	0.84890
Adagrad	0.01	0.9532	0.8131



Fig 20. Adam Optimizer Training

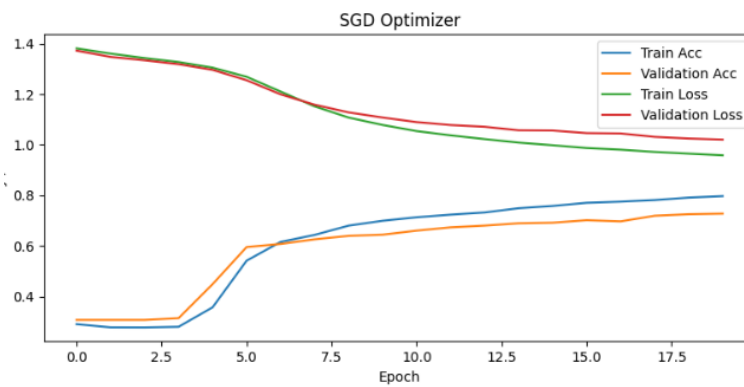


Fig 21. SGD Optimiser Training

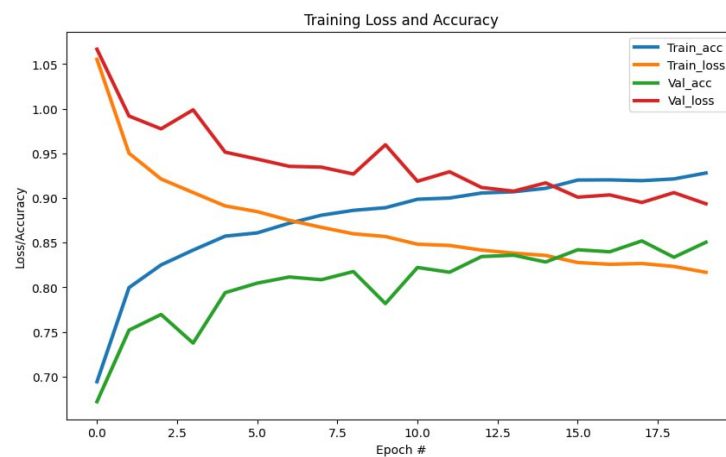


Fig 22. RMSprop Optimiser Training

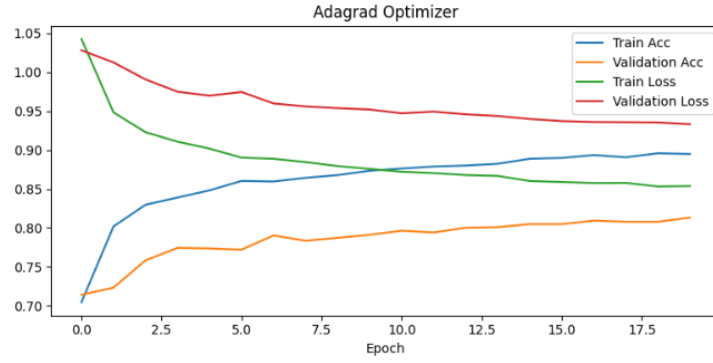


Fig 23. Adagrad Optimiser Training

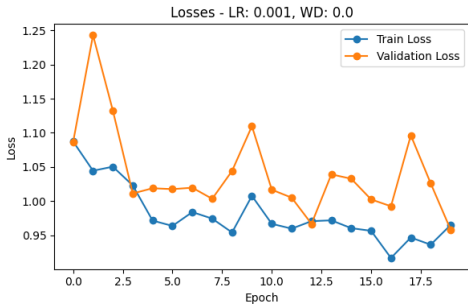
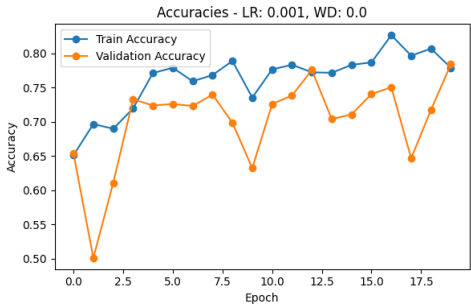
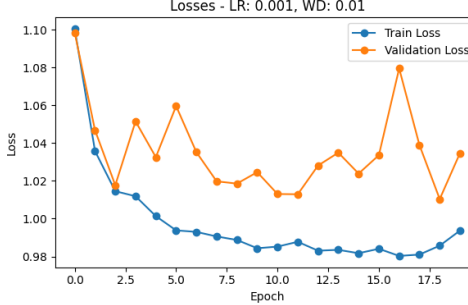
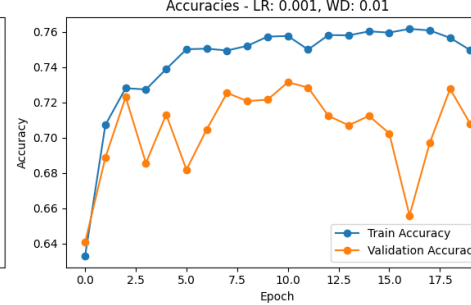
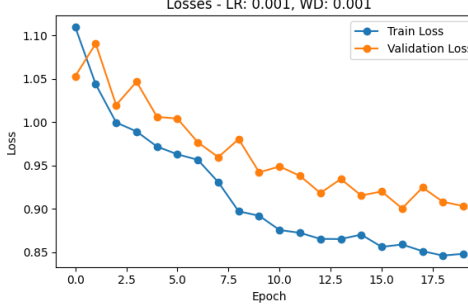
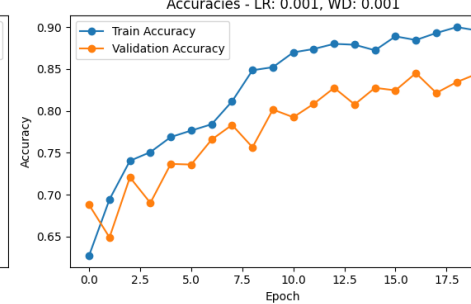
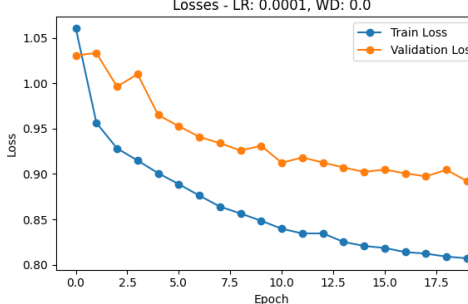
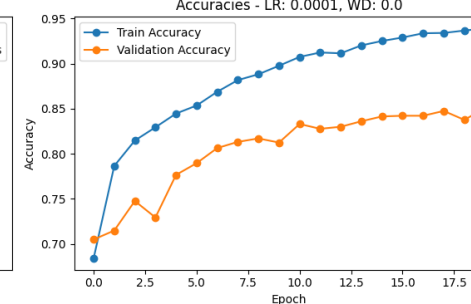
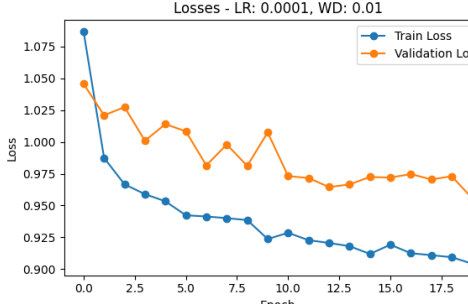
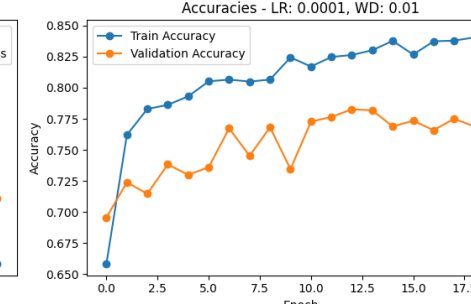
After comprehensive testing with various optimizers, including Adam, RMSprop, Adagrad, and SGD, Adam and RMSprop emerged as top performers with the highest validation accuracy of 83.68% and 84.89% respectively. Adagrad trailed behind with a validation accuracy of 81.31%, while SGD exhibited the lowest accuracy at the same percentage. Despite the variations in performance, Adam remains the preferred optimizer due to its consistently high accuracy and robust performance across various scenarios. Notably, it exhibited the lowest average validation loss compared to RMSprop with high validation accuracy. Its adaptive learning rate mechanism, combined with momentum optimization, facilitates efficient convergence, resulting in superior results when compared to other optimizers like RMSprop, Adagrad, and SGD.

These results underscore the significance of optimizer selection in influencing the performance of our CNN model. Furthermore, these results are also backed up by research from Cornell University, which has shown that Adam has demonstrated superior experimental performance over all the other optimizers such as AdaGrad, SGD and RMSProp (Ajagekar, 2021). This type of optimizer is useful for large datasets, as it is a combination of Momentum and RMSP optimization algorithms, making it straightforward, easy to use, and requiring less memory. Additionally, Adam combines the advantages and benefits of SGDM and RMSProp, utilizing momentum from SGDM and scaling from RMSProp, making it computationally efficient and requiring minimal memory. Moreover, experts suggest that Adam learns all patterns, including noise in the train set, thereby enabling fast convergence (Aremu, 2023). Another research published in Geek Culture has also proved through experimental results that fine-tuned Adam is always better than other optimizers such as SGD and RMSProp (Park, 2021).

Hence, by leveraging Adam as our chosen optimizer, we aim to capitalize on its superior accuracy and robust performance, ultimately enhancing the efficacy of our model in tackling our specific task.

Hyperparameter Tuning on Naïve CNN Model

Hyperparameter tuning involves tuning for the different learning rates and weight decay for Adam on the naive CNN model. We experimented with the following hyperparameters, and the following are its outputs.

Learning Rate	Weight Decay	Loss and Accuracy Graphs	
0.001	0	 	
0.001	0.01	 	
0.001	0.001	 	
0.0001	0	 	
0.0001	0.01	 	

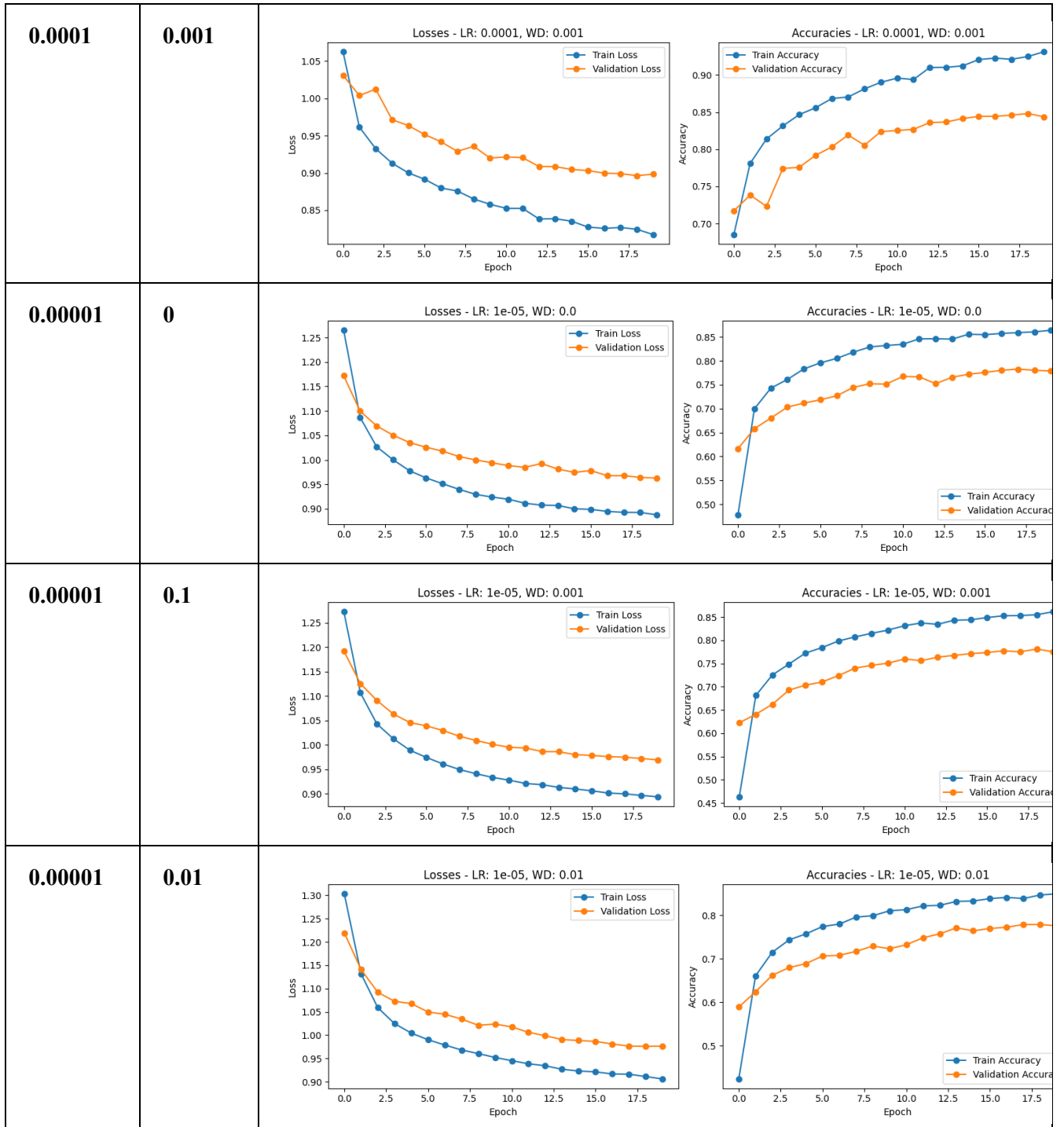


Fig 24. Loss and Accuracy Graph for Hyperparameter Tuning

The trend we noticed is that the lower the learning rate, the lesser the model overfits but the lower the accuracy. The lower the weight decay, the higher the accuracy. Using code to evaluate, the best combination is **Learning Rate = 0.0001 with Weight Decay = 0.001**. The **best test accuracy achieved = 0.8543**, which is higher than the original naïve model.

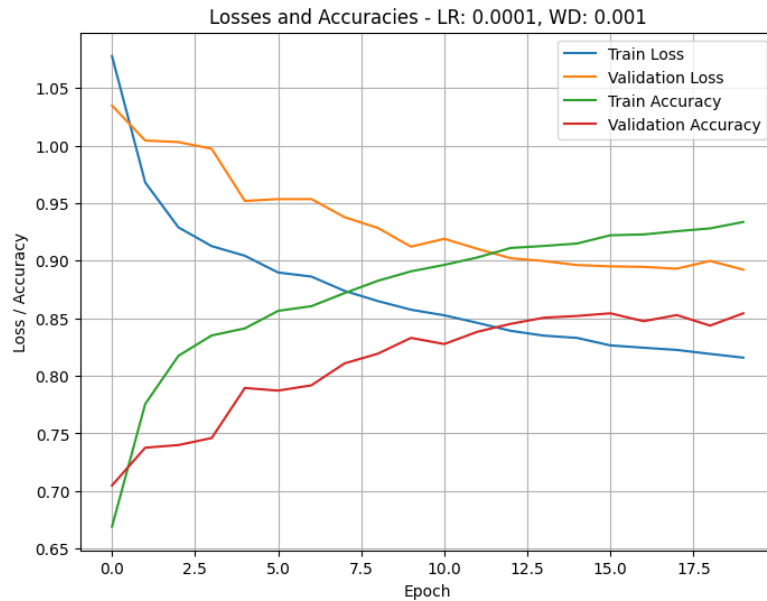


Fig 25. Training using best LR and WD

The model's overfitting tendency and accuracy can be further improved through experimenting on Regularization methods and Data Augmentation.

Data Augmentation on Naïve CNN Model

Data Augmentation is done to increase the robustness of the dataset and introduces diversity into the dataset. This helps the model generalise better to unseen examples. It also reduces the risk of the model memorising the training dataset and prevents it from overfitting.

Using the best optimiser, learning rate and weight decay, data augmentation was done using random rotation, random crop, random flip, random translation, random scaling, colour jittering, gaussian noise and cutout.

```
# Define data augmentation transforms
train_transform = transforms.Compose([
    transforms.RandomRotation(degrees=15),
    transforms.RandomResizedCrop(size=(img_height, img_width), scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.RandomApply([transforms.GaussianBlur(kernel_size=3)], p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Fig 26. Data Augmentation 1

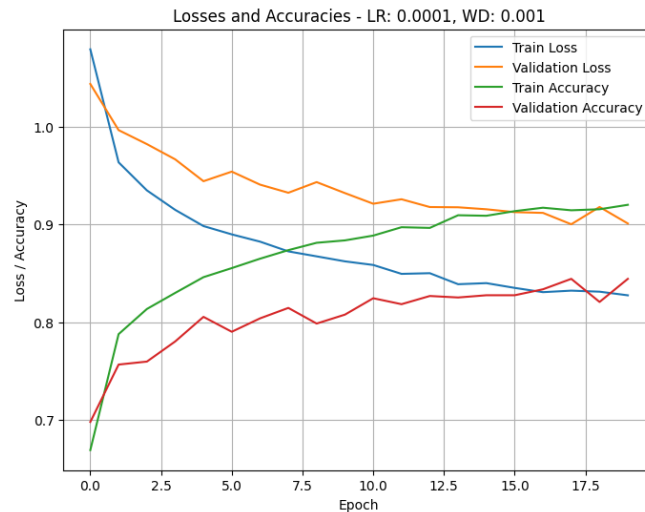


Fig 27. Graph of Data Augmentation Version 1

Using too many data augmentations, might result in the model memorising the training data rather than generalising it. Thus, we experimented on lesser augmentations to see if the model would generalise better and obtain a similar or better accuracy.

```
# Define data augmentation transforms
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(size=(img_height, img_width), scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Fig 28. Data Augmentation Version 2

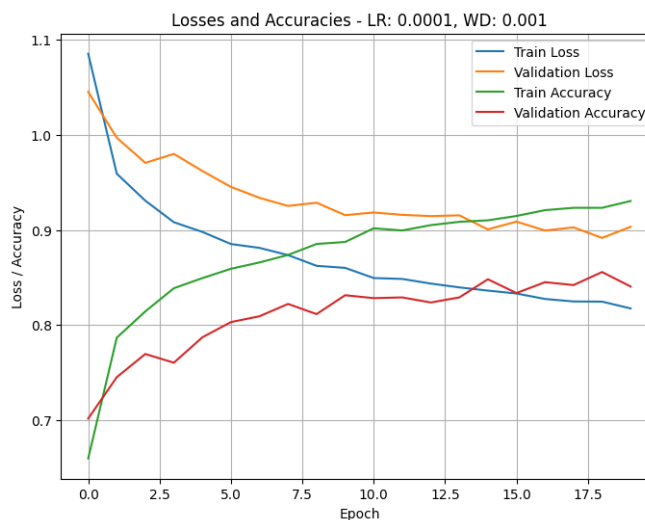


Fig 29. Graph of Data Augmentation Version 2

With lesser augmentations, the model managed to generalise better but performed similarly to base naïve CNN. Thus, further improvements need to be made.

Regularisation Methods on Naïve CNN model

Regularisation methods is done to improve the generalisation of the model by encouraging the model to learn simpler and smoother decision boundaries. It also prevents overfitting by preventing it from fitting the noise in

the training data excessively.

In our model, L2 Regularisation method is experimented using the value best from hyperparameter tuning. However, it is noticeable that the model was not able to generalise well and other methods for regularisation might be needed to perform this.

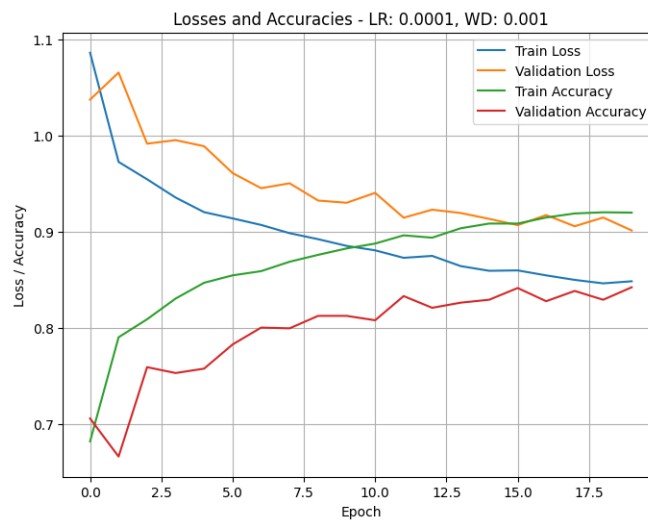


Fig 30. Graph of model with L2 regularisation

Further Improvements

Implementation of a more complex dataset to fine tune the naive CNN model

Dataset complexity, meaning how well it represents the real world, matters more than size for model accuracy especially in the medical domain. Even a small, well-designed dataset can outperform a larger one that doesn't reflect real-world situations (Althnian et al., 2021). Therefore, acquiring such dataset would improve model robustness and accuracy.

Implement deep layers on the CNN Model

As previously mentioned, deeper neural networks with more layers offer improved accuracy by capturing intricate relationships (Bengio, 2013; Khan et al., 2020). Integrating a more comprehensive model with increased depth and width enhances feature recognition and classification robustness. Yet, this advancement requires careful hyperparameter tuning to mitigate overfitting and vanishing gradient problems.

Experimenting on more combinations of hyperparameter tuning using GridSearch

Explore a wider range of hyperparameter values using GridSearch. This allows us to identify potentially hidden optimal configurations for our complex models, leading to significant accuracy gains.

Experimenting on a better combination of data augmentation

Data augmentation bolsters dataset robustness by injecting diversity, aiding the model in generalizing to novel examples and thwarting overfitting by deterring memorization of the training data. While we've touched on a few augmentation techniques, delving into a broader array—like brightness adjustment, sharpening, and inversion—empowers the model to grasp nuanced features and boost accuracy in future endeavours.

Employing early stopping for epoch: To find the optimal balance between model complexity and generalization and prevent overfitting.

Early stopping during model training offers multiple benefits, particularly in time-constrained scenarios and for optimizing hyperparameters. By halting training when validation performance begins to deteriorate, it prevents overfitting, conserves computational resources, and facilitates more efficient time management. This approach ensures that models strike a balance between complexity and generalization, ultimately leading to improved performance and more effective hyperparameter tuning.

Employing different regularization methods to generalise the model.

Employing diverse regularization methods, like L1/L2 penalties and dropout layers (Srivastava et al., 2014), can prevent the complex models from memorizing training data, improve generalizability and overfitting.

Using Initializers (He, Xavier) and Batch Normalisation.

Future advancements can leverage complex models by incorporating proper initialization (He et al., 2015) and Batch Normalization. Initializers like He and Xavier ensure efficient training by setting optimal weight **ranges**.

Conclusion

Brain Tumor Classification Using CNN

Notebook Input Output Logs Comments (8)

In [19]:

```
# Evaluate the model on the training set
train_score = model.evaluate(train_generator)
print(f"Training Loss: {train_score[0]}, Training Accuracy: {train_score[1]}")

# Evaluate the model on the validation set
valid_score = model.evaluate(valid_generator)
print(f"Validation Loss: {valid_score[0]}, Validation Accuracy: {valid_score[1]}")

# Evaluate the model on the test set
test_score = model.evaluate(test_generator)
print(f"Test Loss: {test_score[0]}, Test Accuracy: {test_score[1]}")
```

286/286 [=====] - 149s 521ms/step - loss: 0.2692 - accuracy: 0.9680
Training Loss: 0.2691832482814789, Training Accuracy: 0.9680455327833997
72/72 [=====] - 37s 518ms/step - loss: 0.4603 - accuracy: 0.9274

Fig 31. Best Performing Naïve Model on Kaggle

Through our exploration, we've learned valuable lessons about the efficacy of different model architectures and training techniques. While our naive CNN model achieved a high accuracy of 85%, it distinguished itself by effectively addressing overfitting. In contrast, the Kaggle dataset, while achieving a higher accuracy of 92.7%, may be more susceptible to overfitting. Additionally, our research into transfer learning demonstrated the immense potential of pre-trained models, showcasing a significant leap in accuracy to 99.5% for the same dataset. Our model with further refinement also holds promise for accurately categorizing tumour types, facilitating tailored treatment strategies and improved patient outcomes. As future works on CNN models in the medical field are refined through further fine-tuning, analysis, and potential layer augmentation, its evolution into a pivotal tool for early and accurate brain tumour detection can be actualised. This journey underscores the importance of ongoing research and development in medical imaging and deep learning, driving advancements that have the potential to revolutionise patient care and medical outcomes.

References

- Ajagekar, A. (2021). *Adam*. Cornell University Computational Optimization Open Textbook - Optimization Wiki. <https://optimization.cbe.cornell.edu/index.php?title=Adam>
- Akmalbek Bobomirzaevich Abdusalomov, Mukhriddin Mukhiddinov, & Taeg Keun Whangbo. (2023). *Brain Tumor Detection Based on Deep Learning Approaches and Magnetic Resonance Imaging*. *Cancers*, 15(16), 4172–4172. <https://doi.org/10.3390/cancers15164172>
- Albelwi, S. A. & Department of Computer Science, Faculty of Computing and Information Technology, Industrial Innovation and Robotics Center, University of Tabuk. (2022). *Deep Architecture based on DenseNet-121 Model for Weather Image Recognition*. *International Journal of Advanced Computer Science and Applications: Vol. Vol. 13 (Issue No. 10, pp. 559–560)*. https://thesai.org/Downloads/Volume13No10/Paper_65-Deep_Architecture_based_on_DenseNet_121_Model.pdf
- Althnian, A., AlSaeed, D., Al-Baity, H., Samha, A., Dris, A. B., Alzakari, N., Abou Elwafa, A., & Kurdi, H. (2021, January 15). *Impact of dataset size on classification performance: An empirical evaluation in the medical domain*. MDPI. <https://www.mdpi.com/2076-3417/11/2/796>
- Aremu, T. (2023). *Impact of optimizers in image classifiers*. Towards AI. <https://towardsai.net/p/l/impact-of-optimizers-in-image-classifiers>
- Beginner's Guide to VGG16 implementation in Keras. Built In. (n.d.). <https://builtin.com/machine-learning/vgg16>
- Chen, F., & Tsou, J. Y. (2022). *Assessing the effects of convolutional neural network architectural factors on model performance for remote sensing image classification: An in-depth investigation*. *International Journal of Applied Earth Observation and Geoinformation*, 112, 102865. <https://doi.org/10.1016/j.jag.2022.102865>
- Chollet, F. (2017). *Xception: Deep learning with depthwise separable convolutions*. https://openaccess.thecvf.com/content_cvpr_2017/html/Chollet_Xception_Deep_Learning_CVPR_2017_paper.html
- Convolutional neural Network: benefits, types, and applications*. (2023). Datagen. <https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/#>
- Delobel T, Ayala-Herna'ndez LE, BosqueJJ, Pe'rez-Beteta J, Chulia'n S, Garcí'a-Ferrer M, et al. (2023). *Overcoming chemotherapy resistance in low-grade gliomas: A computational approach*. *PLoS Comput Biol* 19(11): e1011208. <https://doi.org/10.1371/journal.pcbi.1011208>
- Delving deep into rectifiers: Surpassing human-level ... (n.d.-a). https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf
- Dropout: A simple way to prevent neural networks from ... (n.d.-a). <https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>
- GeeksforGeeks, pawangfg. (2024). *VGG-16: CNN model*. <https://www.geeksforgeeks.org/vgg-16-cnn-model/>
- Ilic, I., & Ilic, M. (2023). *International Patterns and trends in the brain cancer incidence and mortality: An observational study based on the global burden of disease*. *Heliyon*, 9(7).

<https://doi.org/10.1016/j.heliyon.2023.e18222>

Jeffrey I. Traylor, MD and John S. Kuo. (n.d.). *Meningiomas – Classifications, risk factors, diagnosis and treatment*. <https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Meningiomas>

Liu, Y., Zhang, L., Hao, Z., Yang, Z., Wang, S., Zhou, X., & Chang, Q. (2022). *An xception model based on residual attention mechanism for the classification of benign and malignant gastric ulcers*. Nature News. <https://www.nature.com/articles/s41598-022-19639-x>

Newman-Toker, D. E., Wang, Z., Zhu, Y., Nassery, N., Saber Tehrani, A. S., Schaffer, A. C., Yu-Moe, C. W., Clemens, G. D., Fanai, M., & Siegal, D. (2020). *Rate of diagnostic errors and serious misdiagnosis-related harms for major vascular events, infections, and cancers: Toward a national incidence estimate using the “Big Three.”* Diagnosis, 8(1), 67–84. <https://doi.org/10.1515/dx-2019-0104>

Park, S. (2021). *A 2021 Guide to improving CNNs-Optimizers: Adam vs SGD*. Medium. <https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008>

Pituitary Tumours. (n.d.). <https://www.singhealth.com.sg/patient-care/conditions-treatments/pituitary-tumours>

Radwan, N. (2019) *Leveraging Sparse and Dense Features for Reliable State Estimation in Urban Environments*. <https://doi.org/10.6094/UNIFR/149856>

Rasheed, S., Rehman, K., & Akash, M. S. H. (2021). *An insight into the risk factors of brain tumors and their therapeutic interventions*. Biomedicine & Pharmacotherapy, 143, 112119. <https://doi.org/10.1016/j.biopha.2021.112119>

Shah, K., Bentley, E., Tyler, A., Richards, K. S., Wright, E., Easterbrook, L., Lee, D., Cleaver, C., Usher, L., Burton, J. E., Pitman, J. K., Bruce, C. B., Edge, D., Lee, M., Nazareth, N., Norwood, D. A., & Moschos, S. A. (2017). *Field-deployable, quantitative, rapid identification of active ebola virus infection in unprocessed blood*. Chem. Sci., 8(11), 7780–7797. <https://doi.org/10.1039/c7sc03281a>

Thakur, R. (2024). *Beginner’s Guide to VGG16 implementation in Keras*. Built In. <https://builtin.com/machine-learning/vgg16>

Vaskovic, J. (2023, November 3). *Normal brain mri*. Kenhub. <https://www.kenhub.com/en/library/anatomy/normal-brain-mri>

Zhao J, Zang F, Huo X and Zheng S. (2023). *Novel approaches targeting ferroptosis in treatment of glioma*. Front Neurol. 14:1292160. <http://dx.doi.org/10.3389/fneur.2023.1292160>

Annexes

Annex 1

Github Link: <https://github.com/ayupermh/Brain-Tumour-Classification.git>