Brain Tumour Classification

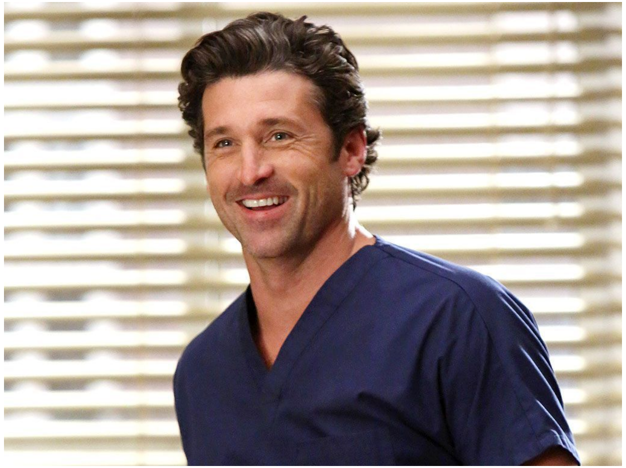
Ayu Permata Halim Mendoza (1006069)

Rio Chan Yu Hoe (1005975)

Parigya Arya (1006096)

Context: You are a part of the team in Grey’s Anatomy

Seattle Grace Hospital / Grey Sloan Memorial Hospital
Year: 2024



Imagine yourself as Derek Shepherd in 2024 (in this case, season 11 did not happen and you did not die yet)

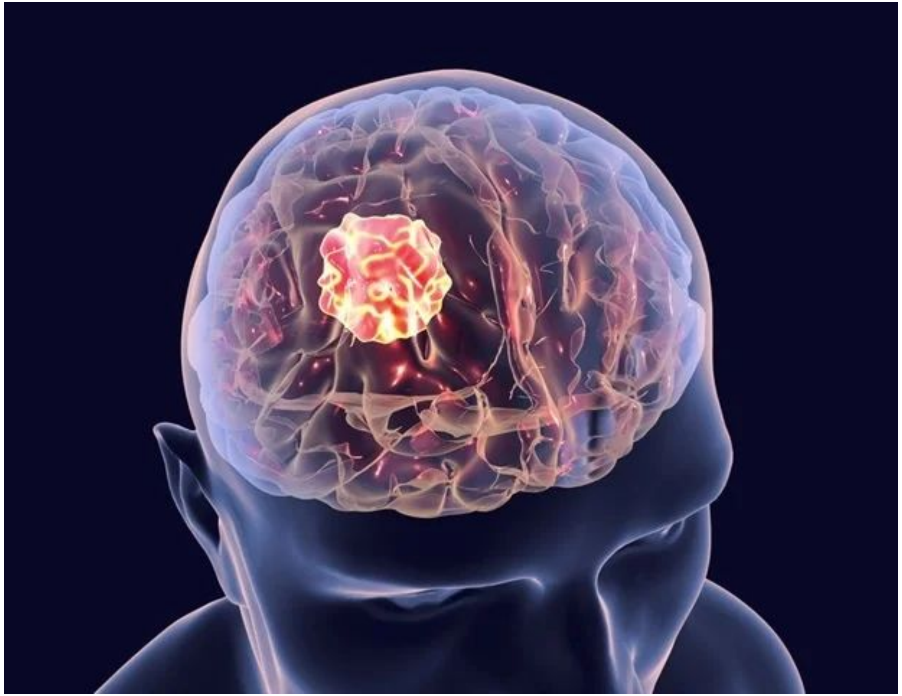
Presentation Flow

1. Problem Scoping, Problem statement
2. Dataset Selection and Preprocessing
3. Base Model Selection and Training
4. Architecture Experimentation
5. Improving our Base Model
6. Further improvements
7. Conclusion

Breakdown of Problem

- Research has shown that the detection and diagnosis of cancers or tumours exhibit an estimated median error rate of 4.4%.

- Detecting tumours manually through MRI scanned images also take doctors a long time & lots of effort.



How Might we **Increase** the **Accuracy** of Brain Tumour **Detection** and **Classification**?

Clustering and Feature Extraction

Deep learning model designed for the classification and identification of brain tumours utilising medical imaging devices, such as MRI

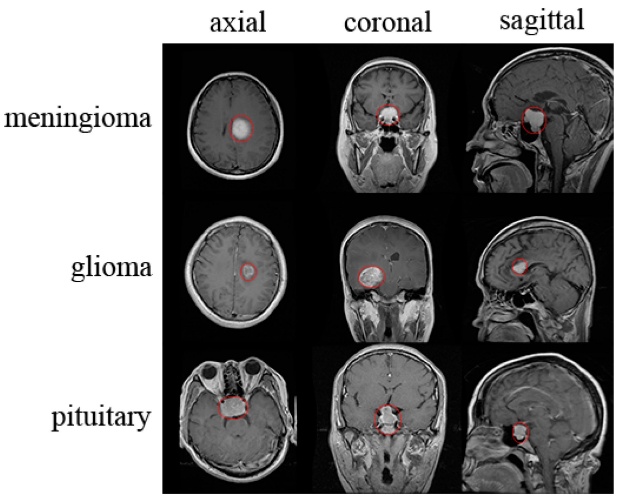
Required Inputs and Outputs

Inputs

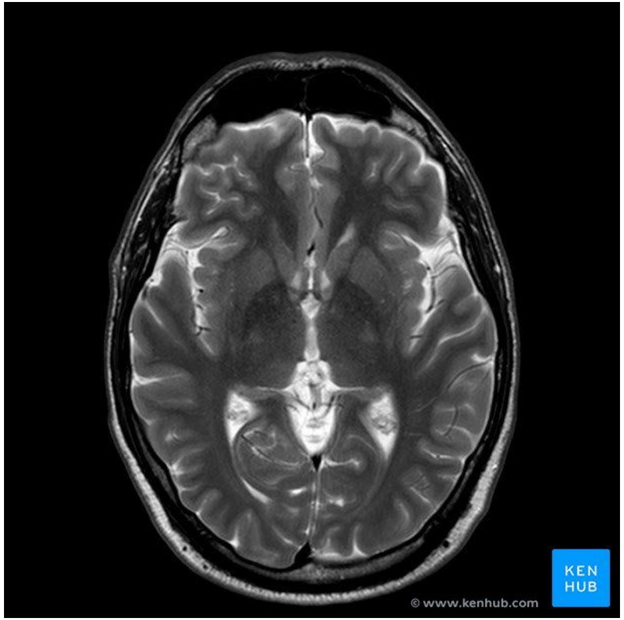
- Images of brain tumours that have been detected through MRI scans
- Dataset chosen - Brain Tumor MRI Dataset from Kaggle

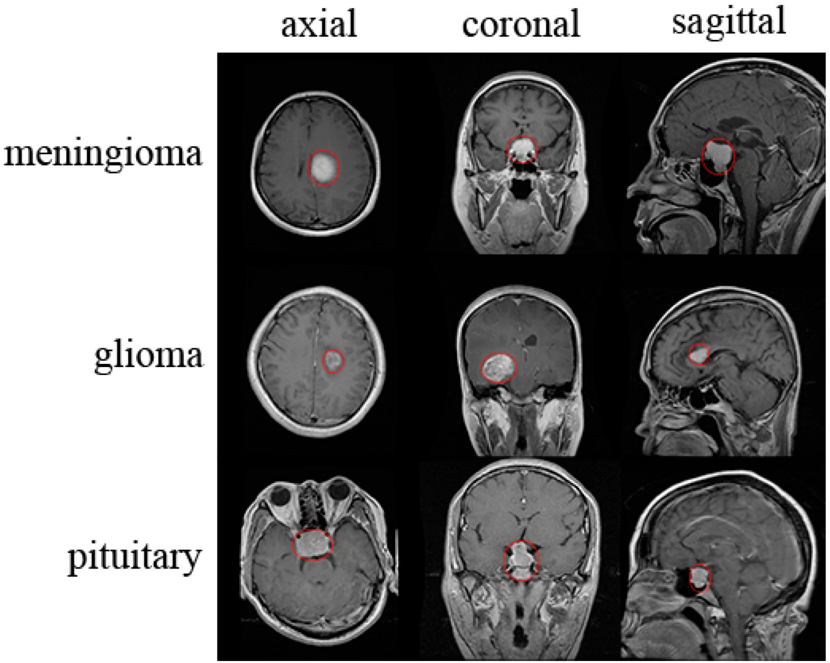
Outputs

- The predicted class or type label of the brain tumour — Glioma, Meningioma, Pituitary Gland Tumour and No Tumour (meaning healthy brain)



Difference between a Healthy Brain and Brain with Tumours





Dataset Preprocessing

Data Transformation

- Transforming image data size and crop in the centre
- Higher resolution will provide more details and thus more robust features can be learned

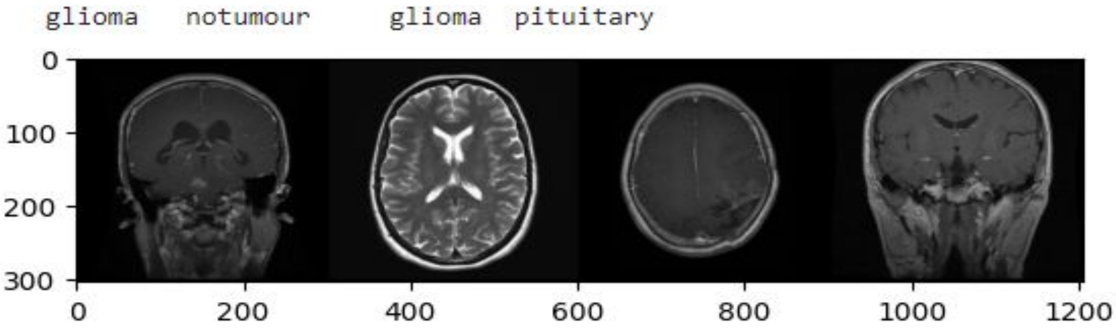
Data Loading

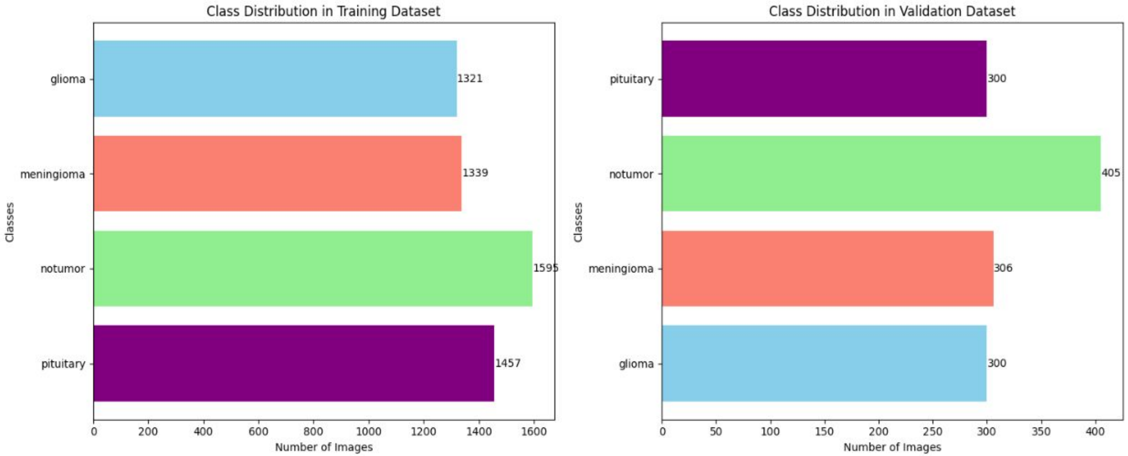
- Dataset split into Training and Testing with indicated batch size

Data Visualisation

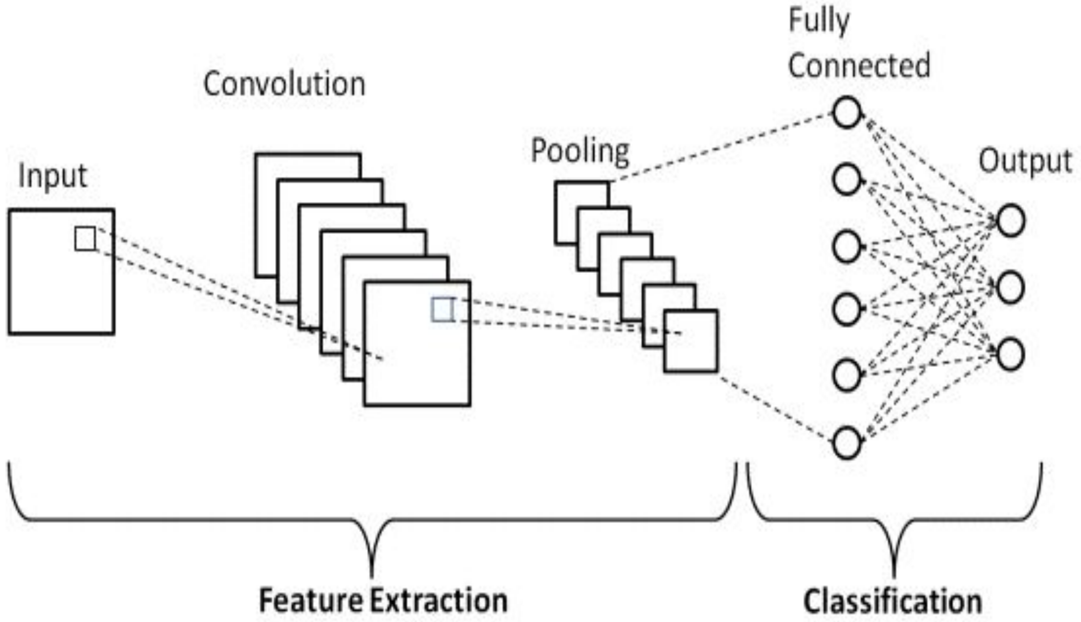
- Visualise cropped images
- Demographics of data splitting

Data Visualisation





Naive CNN



Optimiser: RMSProp
Learning Rate: 0.001
Loss Function: Cross Entropy

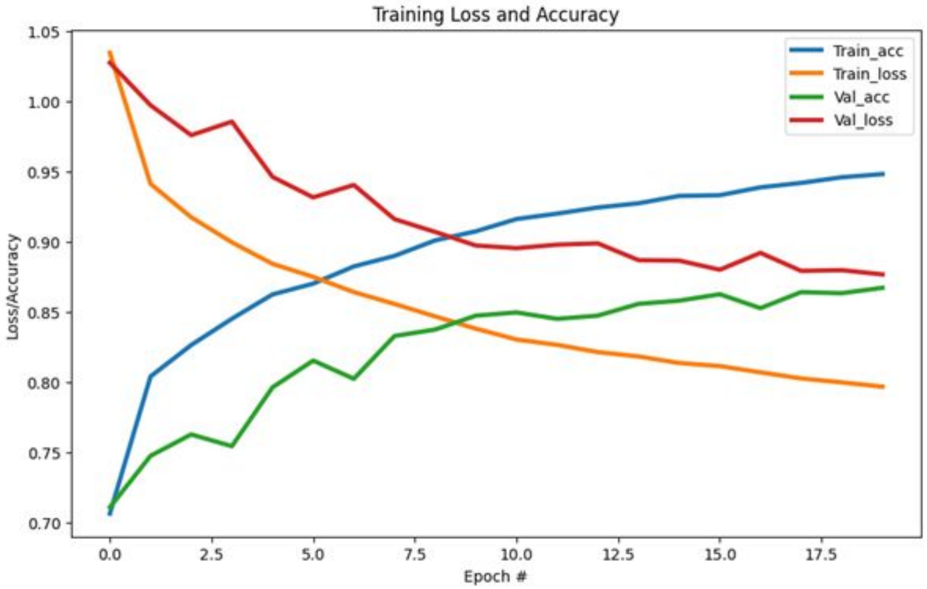
Batch Size and Epoch Size Selection

Batch Size Experimented = [16, 32, 64]

Chosen Batch Size = 16

Epoch Size selected = 20

Training Naive CNN Model



- Model Validation Accuracy = **84%**
- Model is overfitting
- Steps need to be taken to prevent overfitting

Architecture Experimentation

Exploring state-of-the-art image recognition models in healthcare through transfer learning with consistent base model hyperparameters and fine-tuning as necessary

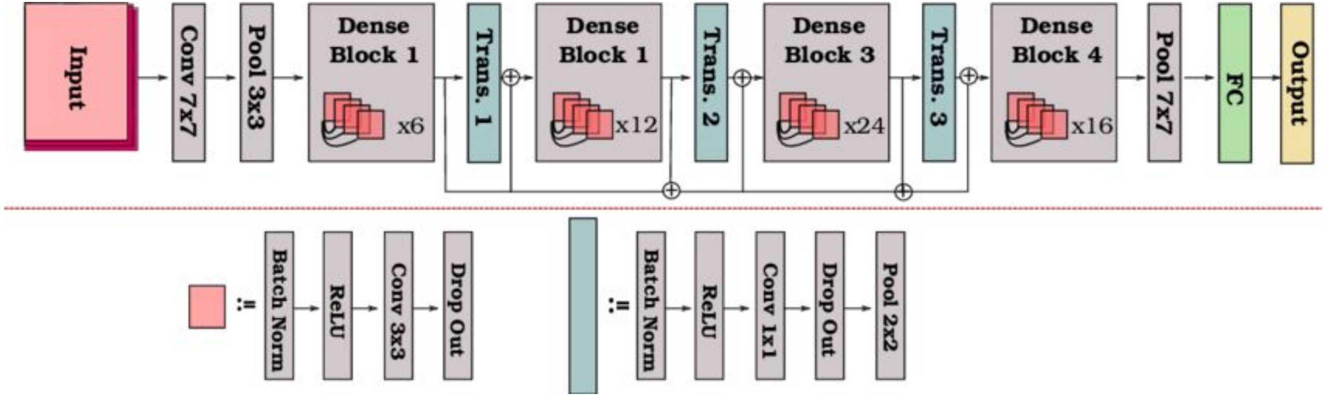
Table 3

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

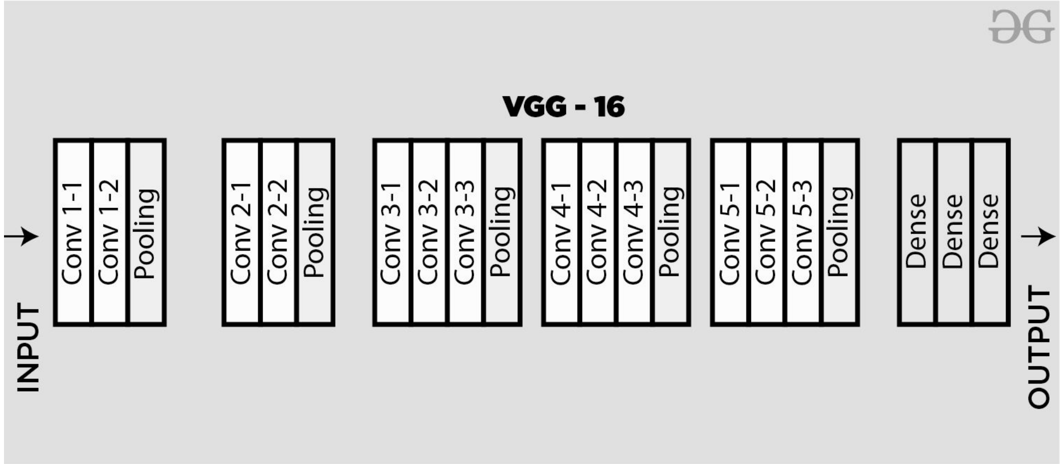
DenseNet121

- Employs batch normalization, ReLU activation, and convolution operations within each layer
- Consists of dense blocks where each layer is connected to all preceding layers
- Allows for efficient information flow and feature reuse, addressing vanishing gradient issues and reducing the number of parameters



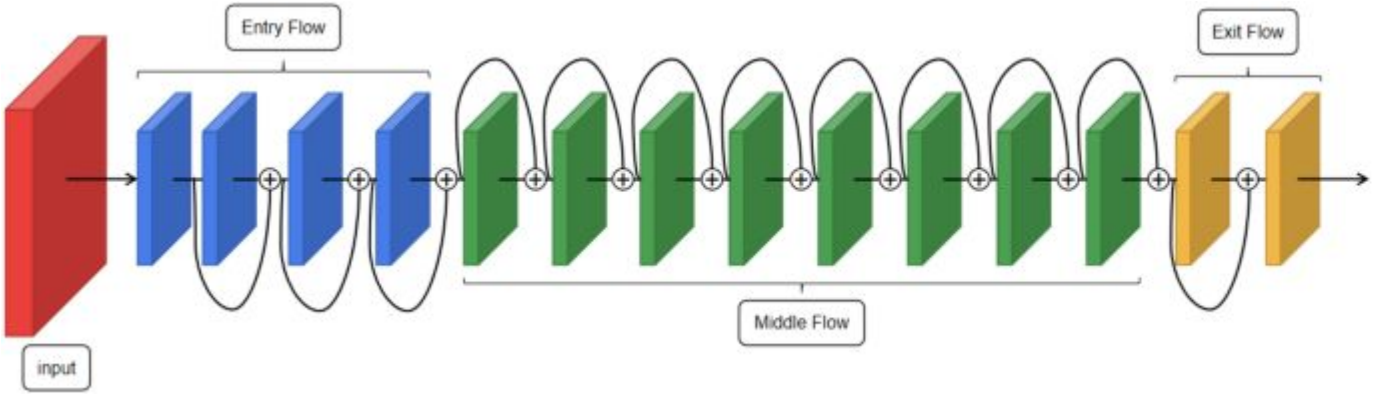
VGG 16

- Employs 3x3 convolutional filters and 2x2 max-pooling layers consistently.
- Its large size—138 million parameters—poses challenges in terms of computational resources and memory requirements.
- Typically operates on images resized to 224x224 pixels



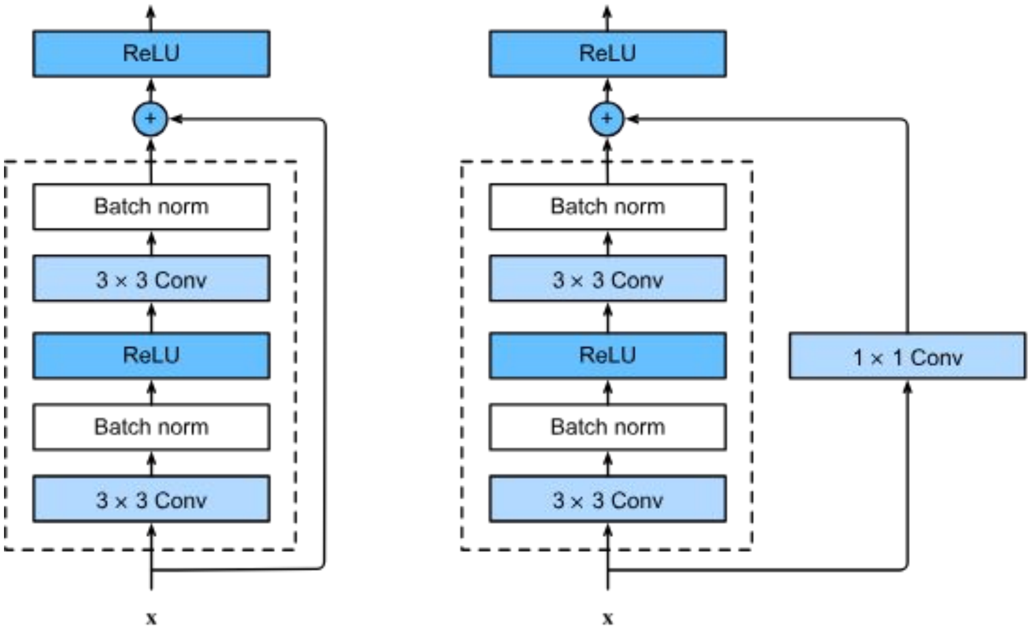
Xception

- Leverages depth-wise separable convolutions, a technique that breaks down standard convolutions for improved efficiency.
- Built with repeated processing blocks, flows through entry, middle, and exit stages
- Lower computational cost compared to traditional architectures



ResNet

- Seminal deep learning model in which the weight layers learn residual functions with reference to the layer inputs
- CNN architecture designed to support hundreds or thousands of convolutional layers
- Highly useful for computer vision applications or image recognition tasks



Results

Model	Train accuracy	Validation Accuracy
Densenet121	85.40%	84.90%
VGG 16	80.20%	80.55%
Xception	91.0%	86.80%
Resnet	88.0%	85.40%

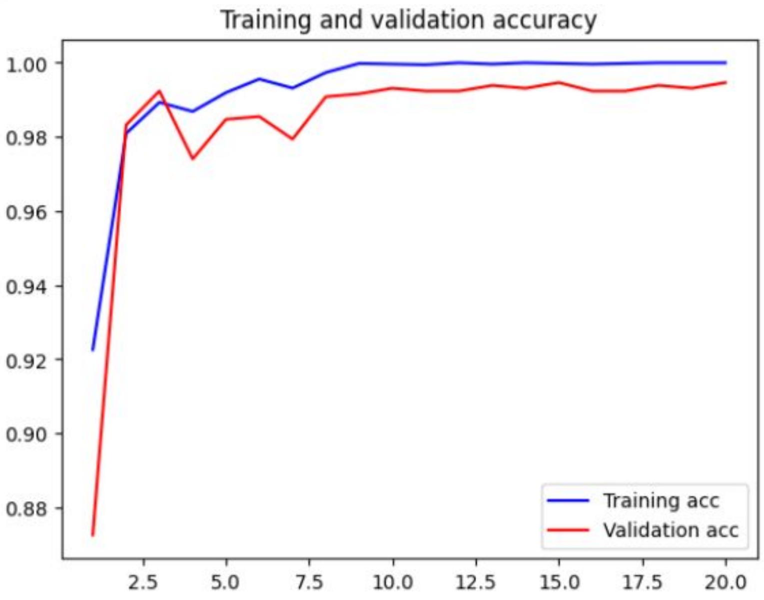
ResNet Tuning

- Employed pre-trained ResNet models (ResNet 50 and ResNet 18)
- Conducted hyperparameter tuning using “Grid Search” to figure out the best combination of hyperparameters

```
# 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
```

ResNet Hyperparameter Tuning Results

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

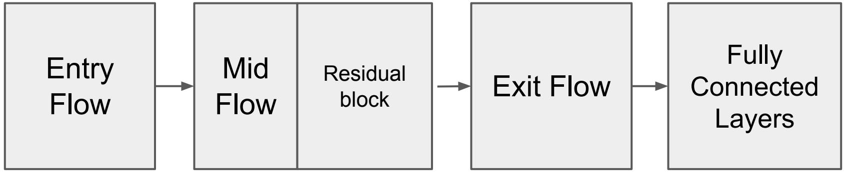



```
Hyperparameters: LR=0.0001, WD=0.001, Model=resnet18
Epoch 1/20, Train Loss: 0.2195, Train Acc: 0.9226, Val Loss: 0.3007, Val Acc: 0.8726
Epoch 2/20, Train Loss: 0.0600, Train Acc: 0.9809, Val Loss: 0.0490, Val Acc: 0.9832
Epoch 3/20, Train Loss: 0.0364, Train Acc: 0.9893, Val Loss: 0.0262, Val Acc: 0.9924
Epoch 4/20, Train Loss: 0.0442, Train Acc: 0.9869, Val Loss: 0.0851, Val Acc: 0.9741
Epoch 5/20, Train Loss: 0.0262, Train Acc: 0.9919, Val Loss: 0.0528, Val Acc: 0.9847
Epoch 6/20, Train Loss: 0.0178, Train Acc: 0.9956, Val Loss: 0.0393, Val Acc: 0.9855
Epoch 7/20, Train Loss: 0.0241, Train Acc: 0.9932, Val Loss: 0.0771, Val Acc: 0.9794
Epoch 8/20, Train Loss: 0.0103, Train Acc: 0.9974, Val Loss: 0.0264, Val Acc: 0.9908
Epoch 9/20, Train Loss: 0.0031, Train Acc: 0.9998, Val Loss: 0.0269, Val Acc: 0.9916
Epoch 10/20, Train Loss: 0.0030, Train Acc: 0.9996, Val Loss: 0.0255, Val Acc: 0.9931
Epoch 11/20, Train Loss: 0.0024, Train Acc: 0.9995, Val Loss: 0.0206, Val Acc: 0.9924
Epoch 12/20, Train Loss: 0.0021, Train Acc: 1.0000, Val Loss: 0.0195, Val Acc: 0.9924
Epoch 13/20, Train Loss: 0.0023, Train Acc: 0.9996, Val Loss: 0.0229, Val Acc: 0.9939
Epoch 14/20, Train Loss: 0.0017, Train Acc: 1.0000, Val Loss: 0.0226, Val Acc: 0.9931
Epoch 15/20, Train Loss: 0.0016, Train Acc: 0.9998, Val Loss: 0.0214, Val Acc: 0.9947
Epoch 16/20, Train Loss: 0.0016, Train Acc: 0.9996, Val Loss: 0.0230, Val Acc: 0.9924
Epoch 17/20, Train Loss: 0.0015, Train Acc: 0.9998, Val Loss: 0.0213, Val Acc: 0.9924
Epoch 18/20, Train Loss: 0.0011, Train Acc: 1.0000, Val Loss: 0.0203, Val Acc: 0.9939
Epoch 19/20, Train Loss: 0.0011, Train Acc: 1.0000, Val Loss: 0.0198, Val Acc: 0.9931
Epoch 20/20, Train Loss: 0.0010, Train Acc: 1.0000, Val Loss: 0.0194, Val Acc: 0.9947
```

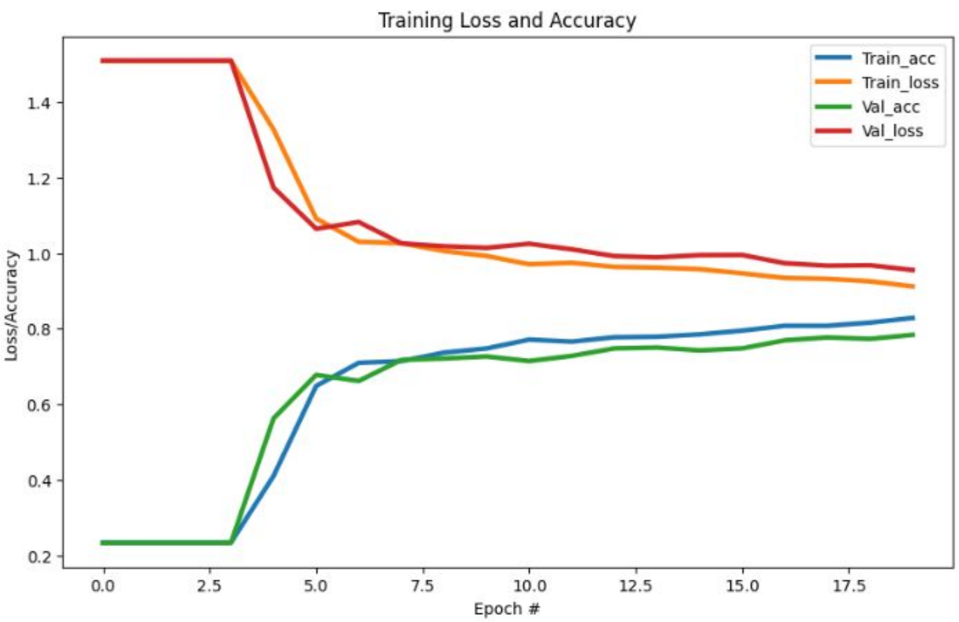
Improving Our Model - Implementation of Elements of Xception and Resnet into Base Model

```python
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)
)
```

- Adopted the residual blocks from Resnet model
- Flow sequence from Xception



Improving Our Model - Implementation of Elements of Xception and Resnet into Base Model



Final training accuracy: 0.8287815126050421
Final validation accuracy: 0.7841342486651411

- Excessive depth for simple dataset, leads to insufficiently trained features.
- Suboptimal learning rates might have hindered convergence, resulting in sluggish and ineffective training processes

Improving Our Model - Experimenting with Different Optimisers

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

'Adam': optim.Adam(model.parameters(), lr=0.001)

'SGD': optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

'RMSprop': optim.RMSprop(model.parameters(), lr=0.001)

'Adagrad': optim.Adagrad(model.parameters(), lr=0.01)

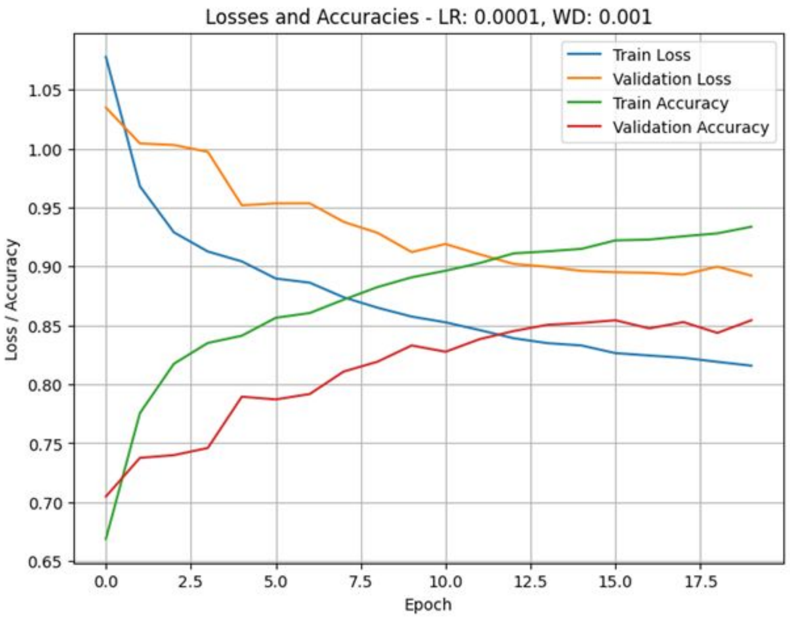
Why ADAM?

- Research by Cornell University has shown that Adam has demonstrated superior experimental performance over all the other optimizers such as AdaGrad, SGD, RMSP, etc in DNN. This type of optimizer is useful for large datasets.
- Adam is a combination of Momentum and RMSP optimization algorithms, and hence it is the most straightforward, easy to use optimiser that requires less memory.

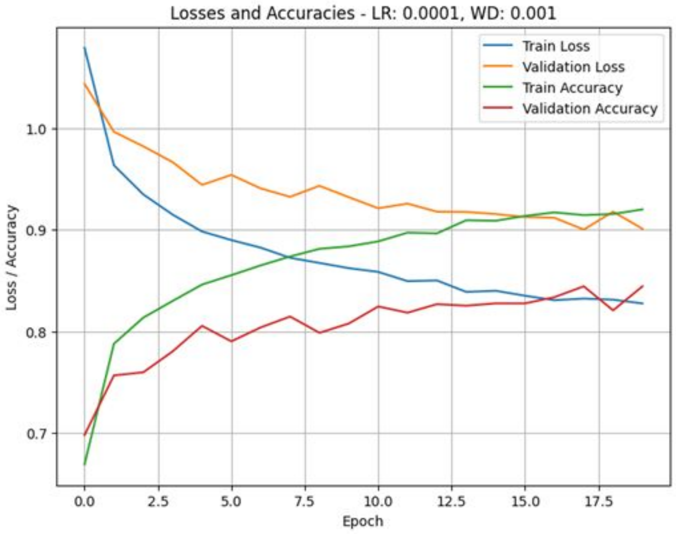
Hyperparameter Tuning

- Experimented with different learning rates and weight decay using Gridsearch
- Learning Rate = [0.001, 0.0001, 0.00001]
- Weight Decay = [0.0, 0.01, 0.001]

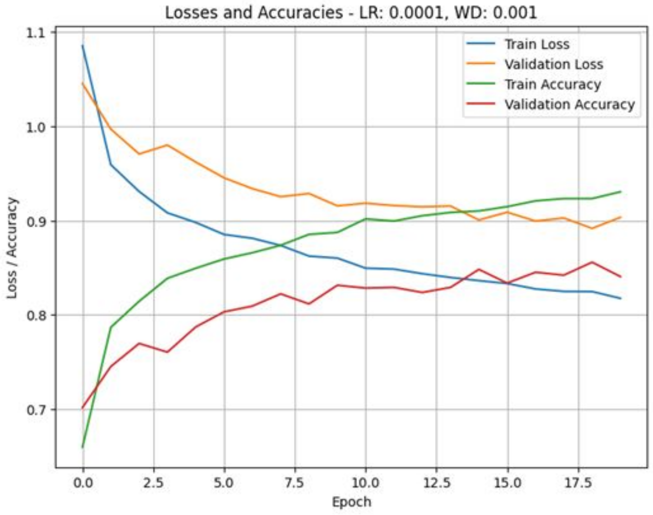
Best Result:
LR = 0.0001, WD = 0.001
Best Val Accuracy = 0.8543



Data Augmentation




```python
# 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])
])
```




```python
# 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])
])
```

Future Improvements

- Implementation of a more complex dataset to fine tune the naive CNN model
- Implement deep layers on the CNN model
- Experimenting on more combinations of hyperparameter tuning using Gridsearch
- Experimenting on a better combination of data augmentation
- Employing early stopping for epoch : To find the optimal balance between model complexity and generalization and prevent overfitting.
- Employing different regularization methods to generalise the model
- Using Initializers (He,Xavier) and Batch Normalisation

Kaggle Dataset’s Training

Brain Tumor Classification Using CNN

▲ 35

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: 0.9680455327033997
72/72 [==============================] - 37s 518ms/step - loss: 0.4603 - accuracy: 0.9274
```

Conclusion

- Kaggle Dataset's Naive CNN Highest Accuracy = **92.7%**
- Our naive CNN Highest Accuracy = **85% with reduced overfitting**
- Training using Pre-Trained Models = **99.5%**

There is much higher accuracy using pre-trained models.

Naive CNN accuracy can be further improved using tuning and further analysing and addition of layers.

References

https://www.mdpi.com/2076-3417/10/6/1999
https://scisynopsisconferences.com/neurology/2023/sessions/neuro-oncology-and-brain-tumors
https://d2l.ai/chapter_convolutional-modern/resnet.html
https://optimization.cbe.cornell.edu/index.php?title=Adam
https://home.adelphi.edu/~kr21836/location.html
https://people.com/tv/greys-anatomys-derek-shepherd-patrick-dempseys-best-mcdreamy-moments/
https://www.mdpi.com/1424-8220/22/12/4324
https://doi.org/10.3390/cancers15164172
https://link.springer.com/chapter/10.1007/978-981-16-6723-7_24
https://www.nature.com/articles/s41598-022-19639-x
https://thesai.org/Downloads/Volume13No10/Paper_65-Deep_Architecture_based_on_DenseNet_121_Model.pdf