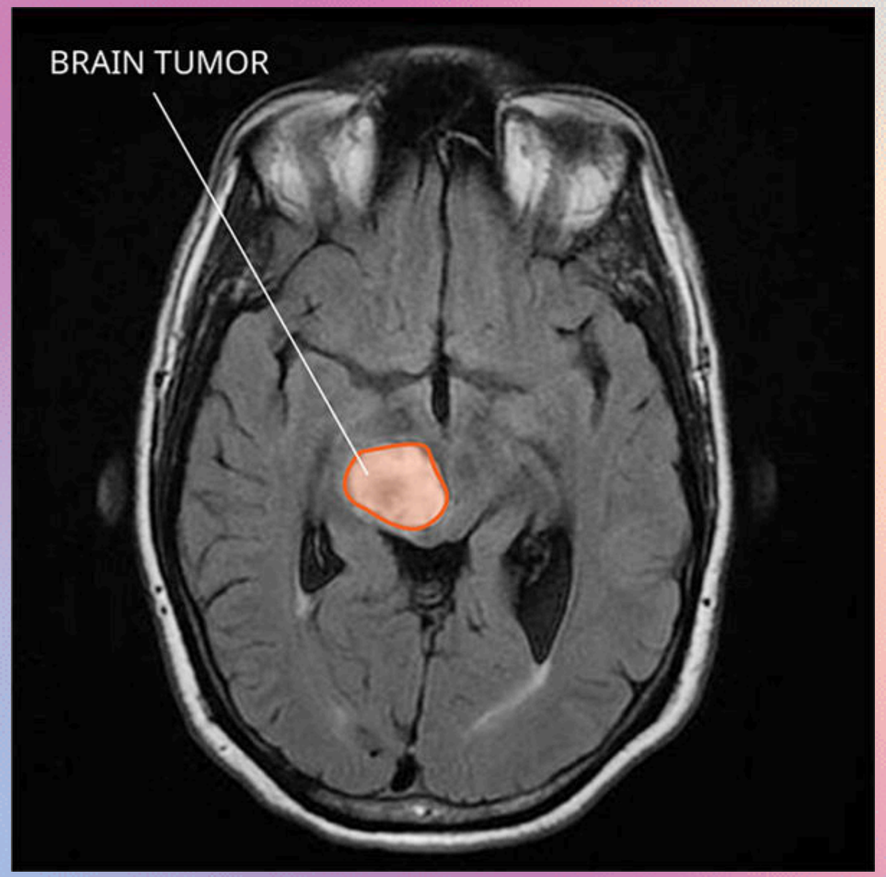
Brain Tumor Classification

MACHINE LEARNING PROJECT
UE21EC352B

NANDHIKA PRAVEEN
PES1UG21EC160

OBJECTIVE



The main objective of brain tumor classification using MRI (Magnetic Resonance Imaging) data is to accurately differentiate between different types of brain tumors based on the features extracted from MRI images.

This classification can help in identifying the appropriate treatment for the patient and can also aid in predicting the patient's prognosis.

DATASETS USED

1) https://github.com/sartajbhuvaji/brain-tumor-classification-dataset

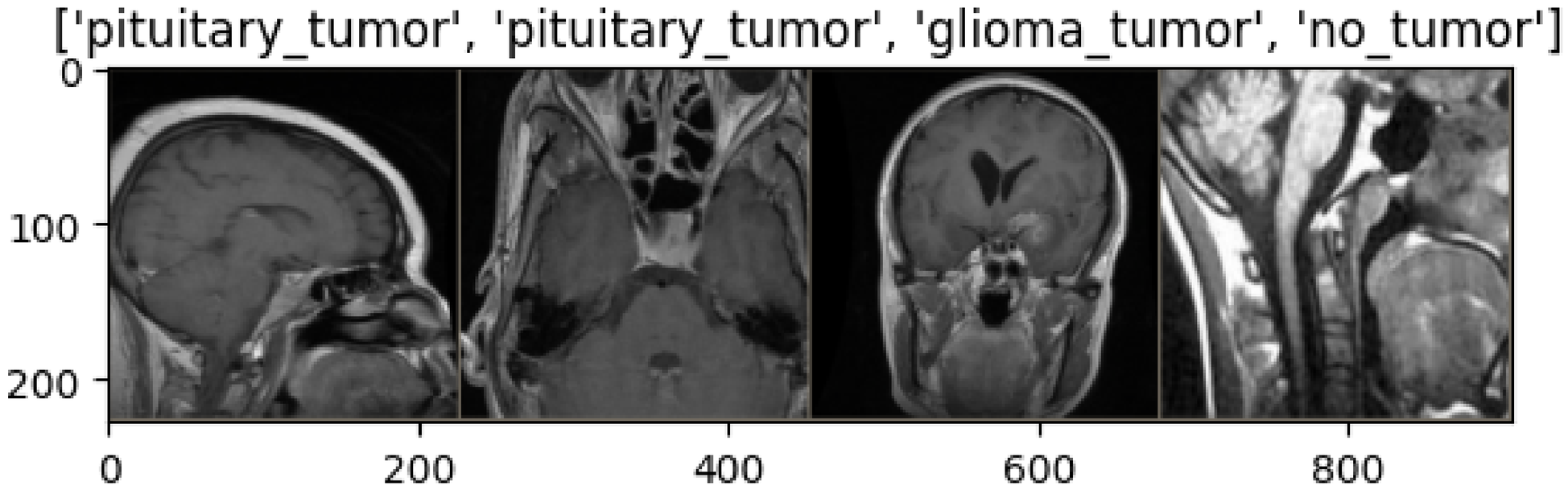
2) https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset

3) https://www.kaggle.com/datasets/mohammadhossein77/brain-tumors-dataset

These datasets contain MRI images and are classified into 4 types of tumors:

- Glioma tumor
- Meningioma tumor
- Pituitary tumor
- No tumor

SAMPLE MRI IMAGES



ALGORITHM USED

The algorithm used in the used code is a common approach for training neural networks known as "Transfer Learning" with the addition of "Fine-tuning". Here's a breakdown of the key components:

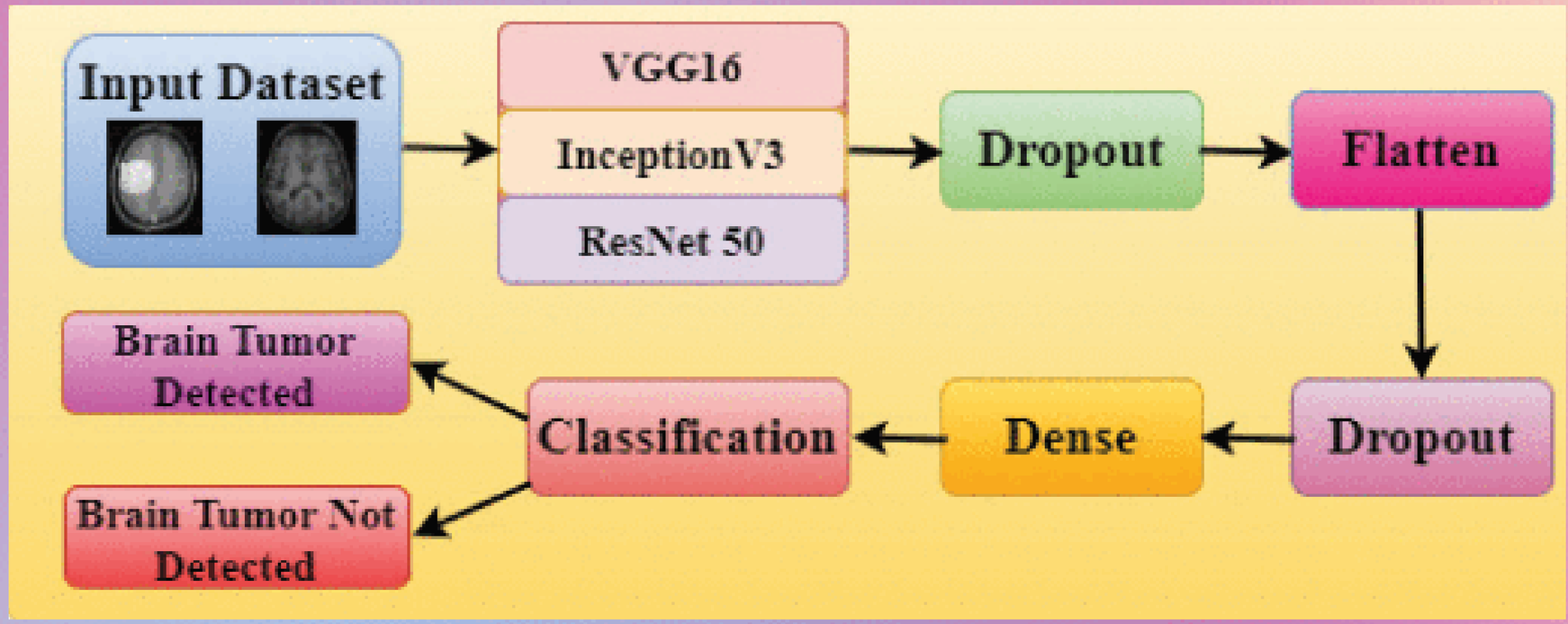
1. **Transfer Learning:** This involves using a pre-trained neural network model (in this case, likely one of the models provided by torchvision) that has been trained on a large dataset, such as ImageNet. Instead of training a model from scratch, which can be computationally expensive and requires a large amount of labeled data, transfer learning allows us to leverage the knowledge (learned features) of the pre-trained model and apply it to our specific task, which is brain tumor classification from MRI images.
2. **Fine-tuning**: Fine-tuning is a technique used in transfer learning where you take a pre-trained model and further train it on your specific dataset. This allows the model to adapt its learned features to the particular characteristics of the new dataset.
3. **Convolutional Neural Networks (CNNs):** CNNs are a type of deep learning algorithm well-suited for image classification tasks. They consist of multiple layers of convolutional and pooling operations, followed by fully connected layers.

MODEL USED

ResNet stands for Residual Network and is a specific type of convolutional neural network (CNN). ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet-50 has an architecture based on the model depicted above, but with one important difference. The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a “bottleneck”, which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer. It uses a stack of three layers rather than two layers.


Input Dataset
VGG16
InceptionV3
ResNet 50
Dropout
Flatten
Dropout
Dense
Classification
Brain Tumor Detected
Brain Tumor Not Detected

MODELS USED IN REFERENCE PAPER

VGG16

As the name suggests, VGG16 is a convolutional neural network that has 16 layers of depth. Trained using over a million images on the ImageNet database, a pre-trained model of the Network is available. The pre-trained Network can categorize photos into 1000 objects, including several animals, a keyboard, a mouse, and a pencil. The Network has therefore acquired a rich classification model for various images. The Network accepts images with a resolution of 224x224.

InceptionV3

- InceptionV3 is a CNN consisting of 48 layers. It was developed as a plugin for GoogleNet to facilitate detection, classification, and picture evaluation.
- It was created to enable deeper networks without permitting the number of parameters to become unfeasibly large; it contains “under 25 million parameters,” as opposed to 60 million for AlexNet.
- The Network accepts images with a resolution of 299x299.

ResNet50

- ResNet stands for Residual Network, a deep CNN architecture that is 50 layers deep.
- It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015. ResNet forms networks based on residual blocks consisting of two or more convolutional layers, batch normalization, and activation functions.
- The Network accepts images with a resolution of 224x224.

TABLE II CNN Pre-Trained Models

Model name	Year	No. of Layers	Input Image Size
VGG16	2014	16	224*224
InceptionV3	2015	48	299*299
ResNet50	2015	50	224*224

IMPLEMENTATION AND IMPORTING NECESSARY LIBRARIES

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
from PIL import Image
from tempfile import TemporaryDirectory

cudnn.benchmark = True
```

DATA LOADING AND AUGMENTATION

```
# Data augmentation and normalization for training and normalization for testing

data_transforms = {
 'Training': transforms.Compose([
 transforms.RandomResizedCrop(224),
 transforms.RandomHorizontalFlip(),
 transforms.ToTensor(),
 transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) ]),
 'Testing': transforms.Compose([
 transforms.Resize(256),
 transforms.CenterCrop(224),
 transforms.ToTensor(),
 transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])]),
}
```

ASSIGNING DATA DIRECTORY

```
#Assigning the data directory
data_dir = '/kaggle/input/brain-tumor-classification-mri'

#Assinging the training and testing folder of the data with os path-joiner
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
 data_transforms[x])
 for x in ['Training', 'Testing']}

#defining the image batch size, initiate image data shuffling and num_workers
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
 shuffle=True, num_workers=4)
 for x in ['Training', 'Testing']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['Training', 'Testing']}

#finding class names from the training data folder
class_names = image_datasets['Training'].classes

#Initiating condition for device selection whether it will be using GPU or CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

TRAINING DATA AND VISUALIZATION

```
#Defining the Visualization function

def imshow(inp, title=None):
 """Display image for Tensor."""
 inp = inp.numpy().transpose((1, 2, 0))
 mean = np.array([0.485, 0.456, 0.406])
 std = np.array([0.229, 0.224, 0.225])
 inp = std * inp + mean
 inp = np.clip(inp, 0, 1)
 plt.imshow(inp)
 if title is not None:
 plt.title(title)
 plt.pause(0.001) # pause a bit so that plots are updated
```

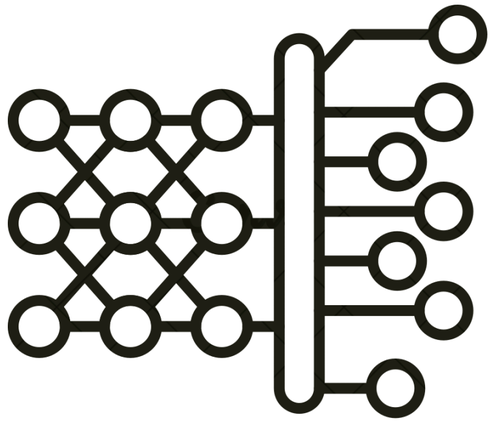
TRAINING DATA AND VISUALIZATION

```
#Defining the Visualization function

def imshow(inp, title=None):
 """Display image for Tensor."""
 inp = inp.numpy().transpose((1, 2, 0))
 mean = np.array([0.485, 0.456, 0.406])
 std = np.array([0.229, 0.224, 0.225])
 inp = std * inp + mean
 inp = np.clip(inp, 0, 1)
 plt.imshow(inp)
 if title is not None:
 plt.title(title)
 plt.pause(0.001) # pause a bit so that plots are updated
```

TRAIN THE MODEL

Let's discuss the function which we have used to ***train the model***. The input parameters which we have used for the train function are:

1. Model: We will input the pre-trained model in here, with the weights of a benchmark dataset (ImageNet-1K/ CIFAR-100/ CIFAR-10).



2. Optimizer: Here, the 'Optimizer' function is mostly depended on two input parameters. Those are:

- Learning rate (lr): The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.
- Stochastic Gradient Descent (SGD): Instead of using all the training data to calculate the gradient per epoch, it uses a randomly selected instance from the training data to estimate the gradient.

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=50):
    since = time.time()
    with TemporaryDirectory() as tempdir:
        best_model_params_path = os.path.join(tempdir, 'best_model_params.pt')
        torch.save(model.state_dict(), best_model_params_path)
        best_acc = 0.0
        for epoch in range(num_epochs):
            print(f'Epoch {epoch}/{num_epochs - 1}')
            print('-' * 10)
            # Each epoch has a training and testing phase
            for phase in ['Training', 'Testing']:
                if phase == 'Training':
                    model.train()  # Set model to training mode
                else:
                    model.eval()   # Set model to evaluate mode
```

```
  running_loss = 0.0
    running_corrects = 0
# Iterate over data.
            for inputs, labels in dataloaders[phase]:
               inputs = inputs.to(device)
               labels = labels.to(device)
               # zero the parameter gradients
               optimizer.zero_grad()
               # forward
               # track history if only in train
               with torch.set_grad_enabled(phase == 'Training'):
                  outputs = model(inputs)
                  _, preds = torch.max(outputs, 1)
                  loss = criterion(outputs, labels)
```

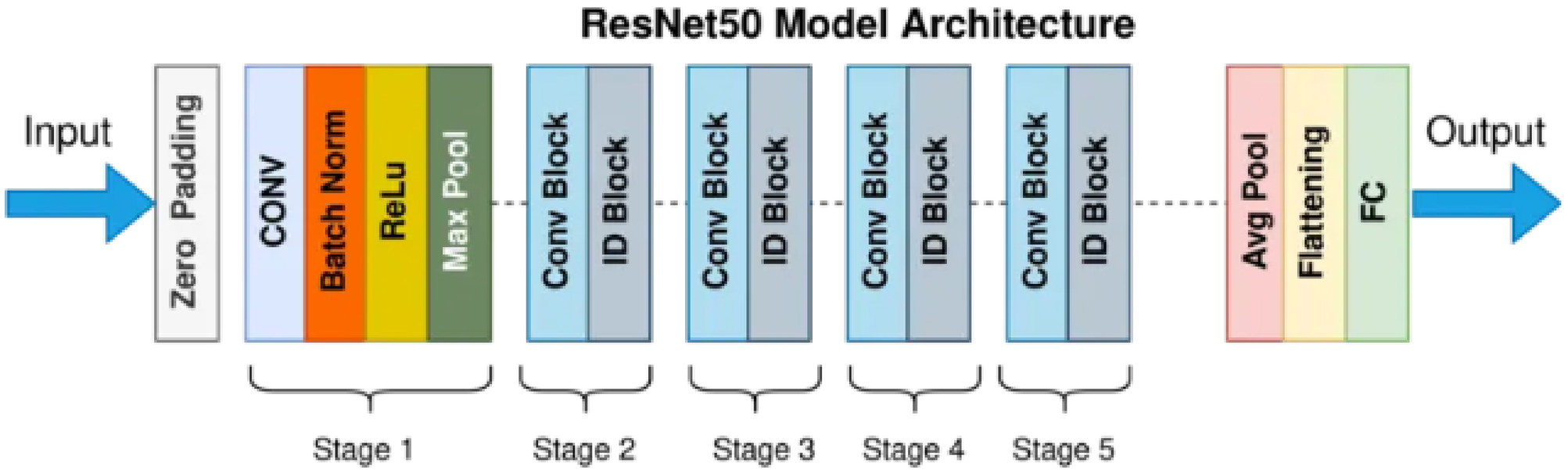
```
# backward + optimize only if in training phase
                if phase == 'Training':
                    loss.backward()
                    optimizer.step()
                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'Training':
                scheduler.step()
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
```

```
# deep copy the model
            if phase == 'Testing' and epoch_acc > best_acc:
                best_acc = epoch_acc
                torch.save(model.state_dict(), best_model_params_path)
            print()
        time_elapsed = time.time() - since
        print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
        print(f'Best val Acc: {best_acc:4f}')
        # load best model weights
        model.load_state_dict(torch.load(best_model_params_path))
    return model
```

```
#Visualizing the model predictions
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()
    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['Testing']):
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
```

```
preds = torch.max(outputs, 1)
        for j in range(inputs.size()[0]):
            images_so_far += 1
            ax = plt.subplot(num_images//2, 2, images_so_far)
            ax.axis('off')
            ax.set_title(f'predicted: {class_names[preds[j]]}')
            imshow(inputs.cpu().data[j])
            if images_so_far == num_images:
                model.train(mode=was_training)
                return
    model.train(mode=was_training)
```

Let's choose to invoke the pre-trained model ***Residual Network- 50 (ResNet50)*** as the core architecture.



Invoking the pre-trained model

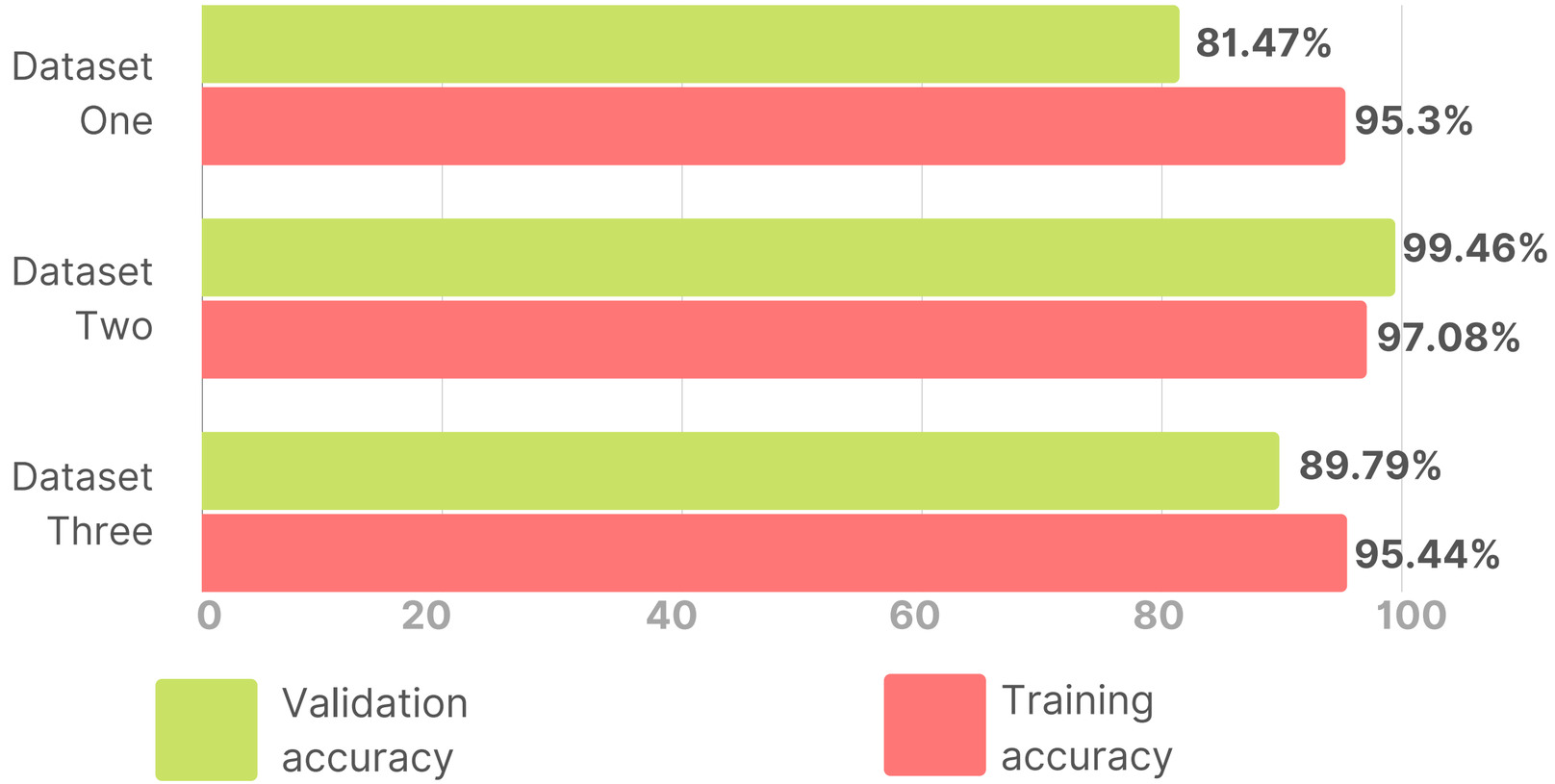
```
# Invoking resnet50 model with the weights of the benchmark dataset of ImageNet-1K-Version1,
#ImageNet is the revolutionary contribution of computer vision researchers from Stanford University and other Institutions
model_ft = models.resnet50(weights='IMAGENET1K_V1')
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, len(class_names))
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
# Setting up the learning rate and momentum
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pt
h" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 226MB/s]
```

print(device)

```
cuda:0
```

START THE TRAINING



COMPARING RESULTS WITH REFERENCE PAPER

TABLE III Result from Comparison Between Transfer Learning Models

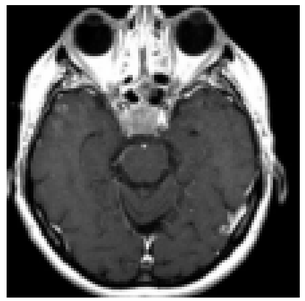
Performance Parameter	VGG16	InceptionV3	ResNet50
Accuracy (in %)	97.24	89.93	94.32
Loss	0.3027	0.2225	0.6273
Validation	91.58	63.86	81.94
Accuracy (in %)			
Validation Loss	1.3502	5.7651	2.0415

MODEL VISUALIZATION

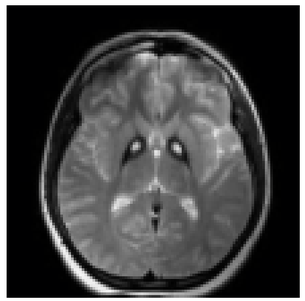
visualize_model(model_ft)

01

predicted: no_tumor



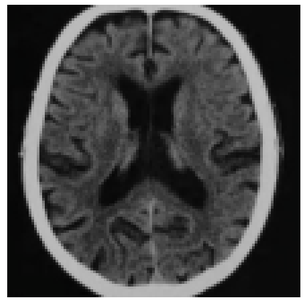
predicted: no_tumor



02

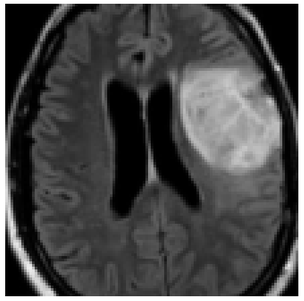
predicted: meningioma_tumor

predicted: no_tumor

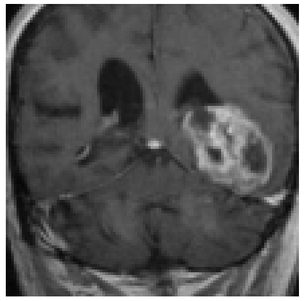


03

predicted: meningioma_tumor



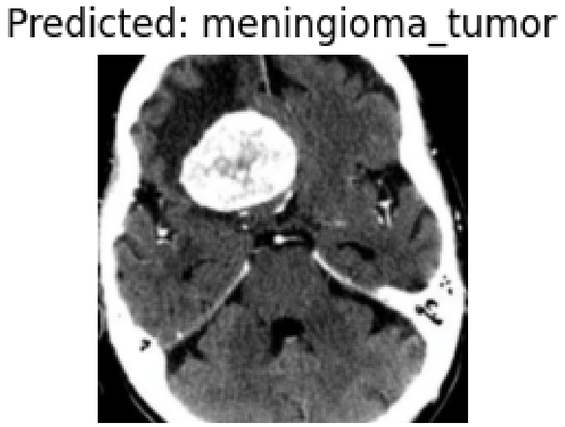
predicted: glioma_tumor



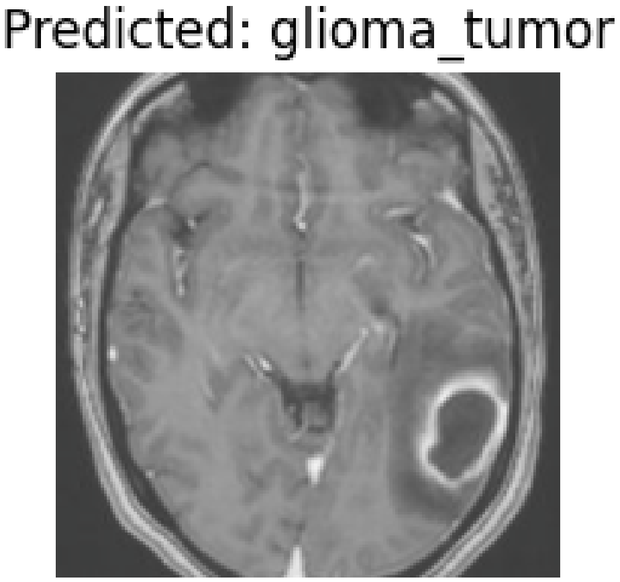
```
def visualize_model_predictions(model,img_path):
    was_training = model.training
    model.eval()

    img = Image.open(img_path)
    img = data_transforms['Testing'](img)
    img = img.unsqueeze(0)
    img = img.to(device)

    with torch.no_grad():
        outputs = model(img)
        _, preds = torch.max(outputs, 1)

        ax = plt.subplot(2,2,1)
        ax.axis('off')
        ax.set_title(f'Predicted: {class_names[preds[0]]}')
        imshow(img.cpu().data[0])


        model.train(mode=was_training)
```

TESTING AND PREDICTING

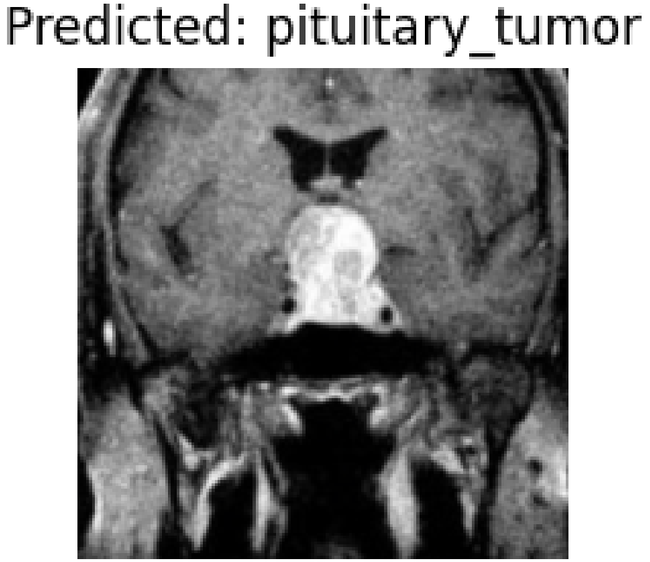
```
#Make an input of image from testing folder, containing an Image of Meningioma Tumor
visualize_model_predictions(model_ft,
img_path='/kaggle/input/brain-tumor-classification-mri/Testing/meningioma_tumor/image(124).jpg')
plt.ioff()
plt.show()
```



```
visualize_model_predictions(
    model_ft,
    img_path='/kaggle/input/brain-tumor-classification-
mri/Testing/glioma_tumor/image(55).jpg')
plt.ioff()
plt.show()
```



```
visualize_model_predictions(
    model_ft,
    img_path='/kaggle/input/brain-tumor-classification-
mri/Testing/pituitary_tumor/image(35).jpg'
)
plt.ioff()
plt.show()
```



```
visualize_model_predictions(
   model_ft,
   img_path='/kaggle/input/brain-tumor-classification-mri/Testing/no_tumor/image(27).jpg')

plt.ioff()
plt.show()
```

NOVELTY IN OUR PROJECT

- In the reference paper, they are classifying whether the MRI images have tumor or not i.e., binary classification. But in our project, we are predicting what kind of tumor the MRI images have.

- The results achieved in the paper:
  Training accuracy - 94.32%

- The results achieved in our project:
  Training accuracy - 95.3%

MAIN CONCEPTS USED

- Convolutional Neural Network
    - Maxpool and Average Pooling
- Dropout and Flattening
- Stochastic Gradient Descent
- Data Augmentation

REFERENCES

R. Pillai, A. Sharma, N. Sharma and R. Gupta, "Brain Tumor Classification using VGG 16, ResNet50, and Inception V3 Transfer Learning Models," 2023 2nd International Conference for Innovation in Technology (INOCON), Bangalore, India, 2023, pp. 1-5, doi: 10.1109/INOCON57975.2023.10101252. keywords: {Deep learning;Technological innovation;Magnetic resonance imaging;Transfer learning;Brain modeling;Time-domain analysis;Time-varying systems;Brain tumor;deep learning;transfer learning models;VGG16;InceptionV3;ResNet50},

THANK YOU

Dropout is a regularization technique for reducing over fitting in neural networks by preventing complex co-adaptations on training data. It is an efficient way of performing model averaging with neural networks.

Skip connections, also known as identity connections, are a key feature of ResNet-50. They allow for the preservation of information from earlier layers, which helps the network to learn better representations of the input data. Skip connections are implemented by adding the output of an earlier layer to the output of a later layer.

Global Average Pooling: Instead of fully connected layers, ResNet typically employs global average pooling. Global average pooling reduces the spatial dimensions of the feature maps to a single value per feature, simplifying the architecture.

Max Pooling: Max pooling layers follow these convolutional layers. They downsample the spatial dimensions of the feature maps, retaining important information and reducing computational load.