Configuration Manual

MSc Research Project
Data Analytics

Rohit Salvi
Student ID: x21208832

School of Computing
National College of Ireland

Supervisor: Paul Stynes

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rohit Salvi
Student ID:	x21208832
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Paul Stynes
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	837
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Rohit Narayan Salvi
Date:	17th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rohit Salvi
x21208832

1 Overview

The configuration manual is the step-by-step guide for setting up the environment, pre-requisites and execution of the research project "Lightweight Deep Learning Framework for Brain Tumour Classification".

2 Hardware/Software Requirements

2.1 Hardware Requirements

The research project was executed on the system with the following hardware configurations.

- **Operating System:** Windows 10 Home Single Language(Version: 22H2).
- **Processor:** Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz.
- **Storage:** 2TB.
- **RAM:** 8GB(Extendable 20.4GB Virtual Memory).
- **Graphical Processing Unit:** NVIDIA GeForce MX150.

2.2 Software Requirements

The software that aided the designing, implementation and execution of the research project are as follows.

- **Development Environment:** Jupyter Notebook(Version: 6.4.12).
- **Programming Language:** Python(Version: 3.10.7).
- **Tools:** Lucidchart and Overleaf.

3 Set Up

3.1 Python

3.1.1 Installing Python

Following are the steps to install Python in the system.

1. Go to offical website of Python[1] and download python source code and installer with version 3.10.7.

2. On the python source code and installer with version 3.10.7 is downloaded. Install Python by running the installer as shown in Figure 1.

Figure 1: Installing Python

3.1.2 Starting and Verifying Python

The following steps will help to verify the availability of Python and Python version in the system.

1. Open the command prompt by searching "Command Prompt" in the Windows search bar.

2. Once the command prompt is opened type `python` and execute it. The version of Python is also displayed on execution of it as shown in Figure 2

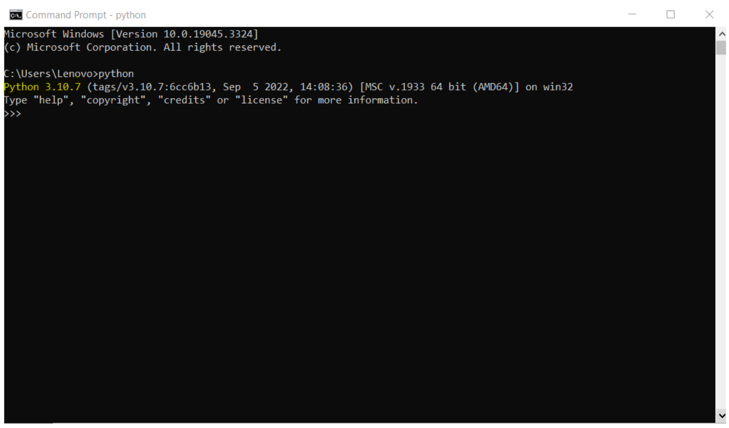

Figure 2: Python and its Version

[1] `https://www.python.org/`

3.2 Jupyter Notebook

3.2.1 Installing and Running Jupyter Notebook

Following are the steps to install and run Jupyter Notebook in the system.

1. Go to offical website of Jupyter[2]. Scroll down to the section of Jupyter Notebook and click on "Install the Notebook" as shown in Figure 3.

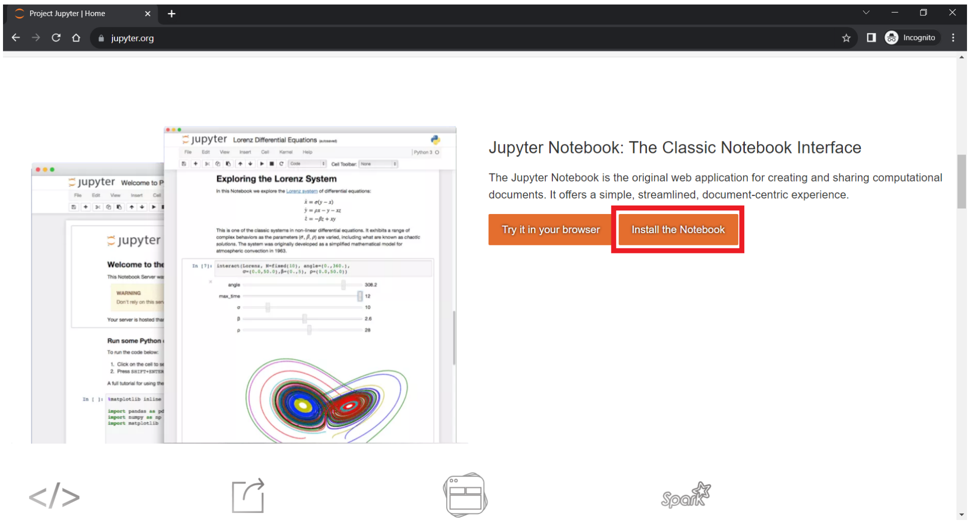

Figure 3: Install Jupyter Notebook

2. After clicking on "Install the Notebook", several ways to install Jupyter will be displayed. Open the command prompt and execute the command as shown in the Figure 4 to install Jupyter Notebook.

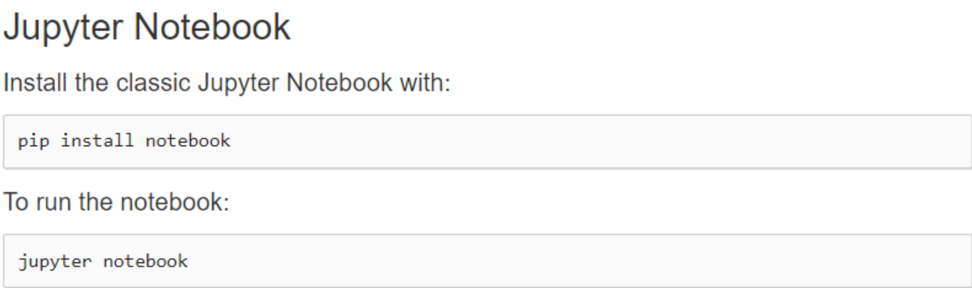

Figure 4: Command to Install and Running Jupyter Notebook

3. On executing the commands mentioned in the above step, a web interface will be popped up in the default browser. Click on "New" and then "Python3" to open the new Jupyter Notebook as shown in Figure 5. Then try executing the simple Python command to ensure the integration and execution of Python in Jupyter Notebook as shown in Figure 6.

[2] `https://jupyter.org/`

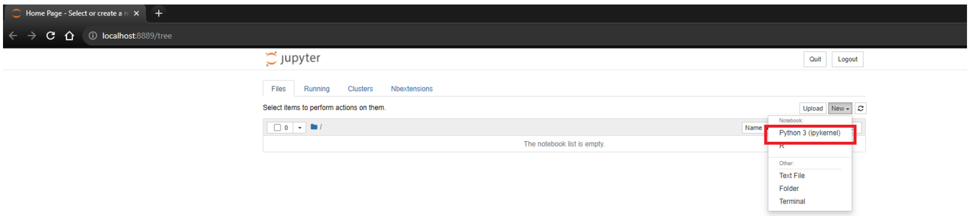

Figure 5: Opening New Jupyter Notebook

Figure 6: Execution of Python Code in Jupyter Notebook

3.3 Data

3.3.1 Data Selection

The data used in the research project is accessible on Kaggle(Nickparvar; 2021). The dataset consists of 7023 '*.jpg*' images of MRIs of 4 different classes of brain tumours.

3.3.2 Importing Data

To import the dataset into the system, navigate to the Brain Tumour MRI Dataset[3] on Kaggle. Hit the download button as shown in Figure 7. Post that select the repository created for the implementation of the research project in the system. Click on download, then download will begin and data will be imported into the system as '*.zip*' file. Extract the data from '*.zip*' file.

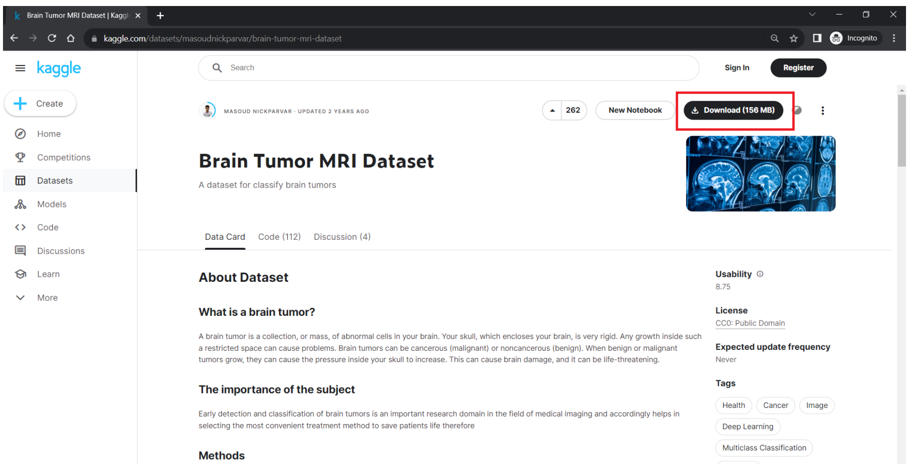

Figure 7: Importing Data from Kaggle

[3]https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset

3.4 Libraries

Following is the list of the packages and libraries required for pre-processing of data, implementation of the model and evaluation of it.

- PyTorch
- Torchvision
- NumPy
- Collections
- Matplotlib
- Pillow
- Scikit-learn
- Seaborn
- Scikit-optimize

3.4.1 Installing Libraries

Install the necessary libraries required for the execution of the research project using the following command.

```
pip install torch
pip install torchvision
pip install matplotlib
pip install Pillow
pip install scikit-learn
pip install scikit-optimize
```

3.4.2 Importing Libraries

Once the required libraries are installed, those libraries are imported, as shown in the Figure 8, for implementation and execution of the research framework.

4 Project Development

4.1 Seed SetUp

For the purpose of reproducibility of research results, set the seed of the environment as shown in the Figure 9.

4.2 Directories SetUp

Directories in which the data is present are set as shown in the Figure 10.
Note: The research data and Jupyter Notebook file are in the same directory.

```python
import os
import torch
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import numpy as np
from collections import Counter
from torch.utils.data import Subset
```
executed in 6.62s, finished 16:37:49 2023-08-13

For Data Visualization

```python
import matplotlib.pyplot as plt
from PIL import Image
```
executed in 1.51s, finished 16:37:50 2023-08-13

For Model

```python
import torch.nn as nn
import torch.optim as optim
```
executed in 11ms, finished 16:37:50 2023-08-13

For Evaluation

```python
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
```
executed in 5.34s, finished 16:37:55 2023-08-13

For Optimization

```python
from skopt.space import Real
from skopt import gp_minimize
```
executed in 597ms, finished 16:37:56 2023-08-13

For Results

```python
from sklearn.metrics import classification_report
```
executed in 12ms, finished 16:37:56 2023-08-13

For K-Fold Validation

```python
from sklearn.model_selection import KFold
```
executed in 11ms, finished 16:37:56 2023-08-13

Figure 8: Importing Libraries

```python
seed = 12
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(seed)
```
executed in 29ms, finished 10:14:54 2023-08-13

Figure 9: Setting Seed

```python
currentDirectory = os.getcwd()
```
executed in 11ms, finished 10:14:54 2023-08-13

```python
baseDirectory = os.path.join(currentDirectory, 'brainTumour')
```
executed in 12ms, finished 10:14:54 2023-08-13

```python
trainingDirectory = os.path.join(baseDirectory, 'Training').replace("\\", "/")
testingDirectory = os.path.join(baseDirectory, 'Testing').replace("\\", "/")
```
executed in 14ms, finished 10:14:54 2023-08-13

Figure 10: Directories SetUp

4.3 Data Loading and Pre-Processing

After setting up the data directories, the data is loaded and pre-processed. The snippet of code for data loading and pr-processing can be seen in the Figure 11.

Data augmentations of Training data

```
trainDataTransform = transforms.Compose([
    transforms.RandomResizedCrop(imageSize),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
])
```

executed in 13ms, finished 10:14:54 2023-08-13

Normalizing Testing Data

```
testDataTransform = transforms.Compose([
    transforms.Resize(imageSize),
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
])
```

executed in 14ms, finished 10:14:54 2023-08-13

Loading Data

```
trainData = ImageFolder(trainingDirectory, transform=trainDataTransform)
testData = ImageFolder(testingDirectory, transform=testDataTransform)
```

executed in 3.72s, finished 10:14:58 2023-08-13

Figure 11: Code for Data Loading and Pre-Processing

4.4 CNN Model

The CNN model of the research project has similar architecture of the model by Soewu et al. (2022). The architecture and code for the implementation of CNN can be seen in the Figure 12.

```
class CNNModel(nn.Module):
    def __init__(self, num_classes):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 16, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(16 * 32 * 32, 256)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 16 * 32 * 32)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

executed in 13ms, finished 10:14:59 2023-08-13

Create the model instance

```
model = CNNModel(num_classes=len(categories))
model.to(device)
```

executed in 75ms, finished 10:14:59 2023-08-13

```
CNNModel(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=16384, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=4, bias=True)
)
```

Figure 12: CNN Model

CNN model is trained with the training data as shown in the Figure 13.

Define the loss function and optimizer

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

executed in 13ms, finished 10:14:59 2023-08-13

Training CNN Model

```
lossValuesOfBaseCNN = []
accuracyValuesOfBaseCNN = []
for epoch in range(epochs):
    model.train()
    runningLoss = 0.0
    correctPredictions = 0
    totalPredictions = 0
    for inputs, labels in trainDataLoader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        runningLoss += loss.item()
        _, predicted = torch.max(outputs, 1)
        totalPredictions += labels.size(0)
        correctPredictions += (predicted == labels).sum().item()

    # Calculate average loss for the epoch
    averageLossOfBaseCNN = runningLoss / len(trainDataLoader)
    lossValuesOfBaseCNN.append(averageLossOfBaseCNN)

    # Calculate accuracy for the epoch
    accuracy = correctPredictions / totalPredictions
    accuracyValuesOfBaseCNN.append(accuracy)

    # Print training loss after each epoch
    print(f"CNN Model - Epoch {epoch+1}/{epochs}, Loss: {averageLossOfBaseCNN}, Accuracy: {accuracy}")
```

executed in 1h 54m 41s, finished 12:09:40 2023-08-13

Figure 13: Training CNN

4.5 Pruning CNN Model

CNN Model is pruned using a magnitude-based weight training technique. The function for pruning the CNN model can be seen in the Figure 14.

Function to prune CNN Model

```
def pruneModel(model, pruningRatio):
    allParameters = []
    for paramName, param in model.named_parameters():
        if 'weight' in paramName:  # Only prune weights, not biases
            allParameters.append((paramName, param.data.view(-1)))

    # Flatten and concatenate all weights
    allWeights = torch.cat([param for _, param in allParameters])

    # Calculate the threshold value for pruning
    numParamsToPrune = int(pruningRatio * len(allWeights))
    threshold = torch.topk(torch.abs(allWeights), numParamsToPrune).values.min()

    # Apply pruning mask to each parameter tensor
    for paramName, param in allParameters:
        mask = torch.abs(param) > threshold
        param.data *= mask.float()

    print(f"Pruning {pruningRatio*100:.2f}% of model parameters.")

    return model
```

executed in 23ms, finished 12:10:23 2023-08-13

```
def getPrunedModel(originalModel, pruningRatio):
    # Create a new model instance
    prunedModel = CNNModel(num_classes=len(categories))
    prunedModel.to(device)

    # Load the state_dict from the original model to the new model
    prunedModel.load_state_dict(originalModel.state_dict())

    # Apply pruning to the new model
    prunedModel = pruneModel(prunedModel, pruningRatio)

    return prunedModel
```

executed in 26ms, finished 12:10:23 2023-08-13

Figure 14: Pruning Function

Later the pruned model is re-trained on the training data.

4.6 Optimal Pruning Ratio

The optimal pruning ratio is identified using the objective function as shown in Figure 15.

Define the objective function to be minimized

```
def objectiveFunction(params):
    pruningRatio = params[0]  # Extract the pruning_ratio from the list of params
    prunedModel = getPrunedModel(model, pruningRatio)

    # Define the loss function and optimizer for the pruned model
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(prunedModel.parameters(), lr=0.001)

    # Training loop for the pruned model
    for epoch in range(epochs):
        prunedModel.train()
        runningLoss = 0.0
        for inputs, labels in trainDataLoader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = prunedModel(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            runningLoss += loss.item()

        # Print training loss after each epoch
        print(f"Model - Epoch {epoch+1}/{epochs}, Loss: {runningLoss/len(trainDataLoader)}")

    prunedModel.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in testDataLoader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = prunedModel(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return -accuracy
```
executed in 25ms, finished 06:30:33 2023-08-10

Define the search space for pruning ratio

```
searchSpace = Real(low = 0.01, high = 0.5, prior = 'log-uniform')
```
executed in 25ms, finished 06:30:33 2023-08-10

Performing Optimization

```
result = gp_minimize(objectiveFunction, dimensions = [searchSpace], n_calls = 10, random_state = seed)
```
executed in 17h 31m 56s, finished 00:02:29 2023-08-11

Figure 15: Objective Function for identifying Optimal Pruning Ratio

Then lightweight deep learning framework is built using the CNN model, pruning functional and optimal pruning ratio.

5 Project Testing

The developed framework is validated with the help of testing data. The code for the testing of the framework is in the Figure 16.

```
optimisedPrunedModel.eval()
correct = 0
total = 0
predictedLabelsOfOptimisedPrunedModel = []
trueLabelsOfOptimisedPrunedModel = []
with torch.no_grad():
    for inputs, labels in testDataLoader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = optimisedPrunedModel(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Save predicted and true labels for calculating metrics
        predictedLabelsOfOptimisedPrunedModel.extend(predicted.cpu().numpy())
        trueLabelsOfOptimisedPrunedModel.extend(labels.cpu().numpy())
```
executed in 38.7s, finished 01:43:01 2023-08-11

Figure 16: Validating the Framework

To validate the robustness of the model, k-fold cross-validation technique is applied as shown in the Figure 17.

```
kFolds = 5
```
executed in 10ms, finished 09:52:16 2023-08-11

```
kf = KFold(n_splits = kFolds, shuffle = True, random_state = seed)
```
executed in 15ms, finished 09:52:17 2023-08-11

```
foldAccuracies = []
```
executed in 20ms, finished 09:52:18 2023-08-11

```
for fold, (train_index, val_index) in enumerate(kf.split(trainData)):
    print(f"Fold {fold+1}/{k_folds}")

    # Create data loaders for current fold
    trainSubset = Subset(trainData, train_index)
    valSubset = Subset(trainData, val_index)

    trainLoader = DataLoader(trainSubset, batch_size=32, shuffle=True, num_workers=4)
    valLoader = DataLoader(valSubset, batch_size=32, shuffle=False, num_workers=4)

    # Train the optimised pruned model on the current fold
    for epoch in range(epochs):
        optimisedPrunedModelForKFold.train()
        runningLoss = 0.0
        for inputs, labels in trainLoader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = optimisedPrunedModelForKFold(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            runningLoss += loss.item()

        # Print training loss after each epoch
        print(f"Optimised Pruned Model - Fold {fold+1} - Epoch {epoch+1}/{epochs}, Loss: {runningLoss/len(trainLoader)}")

    # Evaluate the optimised pruned model on the validation data for the current fold
    optimisedPrunedModelForKFold.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in valLoader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = optimisedPrunedModelForKFold(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    foldAccuracies.append(accuracy)
```
executed in 5h 42m 18s, finished 15:43:07 2023-08-11

Figure 17: K-Fold Cross Validation

The performance metrics of the framework are calculated as shown in the Figure 18, 19, 20 and 21

```
accuracyOfOptimisedPrunedModel = accuracy_score(trueLabelsOfOptimisedPrunedModel, predictedLabelsOfOptimisedPrunedModel)
print(f"Accuracy of Optimised Pruned CNN: {accuracyOfOptimisedPrunedModel*100}")
```
executed in 38ms, finished 01:43:01 2023-08-11

Figure 18: Computing Accuracy

```
confusionMatrixOfOptimisedPrunedModel = confusion_matrix(trueLabelsOfOptimisedPrunedModel, predictedLabelsOfOptimisedPrunedMode
```
executed in 39ms, finished 01:43:01 2023-08-11

```
plt.figure(figsize = (8, 6))
sns.heatmap(confusionMatrixOfOptimisedPrunedModel, annot = True, fmt = "d", cmap = "Blues", xticklabels = categories, yticklabe
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```
executed in 2.52s, finished 01:43:04 2023-08-11

Figure 19: Computing Confusion Matrix

```
for idx, category in enumerate(categories):
    TP = confusionMatrixOfOptimisedPrunedModel[idx, idx]
    FN = np.sum(confusionMatrixOfOptimisedPrunedModel[idx, :]) - TP
    FP = np.sum(confusionMatrixOfOptimisedPrunedModel[:, idx]) - TP
    TN = np.sum(confusionMatrixOfOptimisedPrunedModel) - TP - FN - FP

    sensitivity = TP / (TP + FN)
    specificity = TN / (TN + FP)

    print(f"Class: {category}")
    print(f"Sensitivity: {sensitivity * 100:.2f}%")
    print(f"Specificity: {specificity * 100:.2f}%")
    print()
```
executed in 27ms, finished 01:43:04 2023-08-11

Figure 20: Computing Sensitivity and Specificity

```
plt.plot(range(1, epochs + 1), lossValuesOfOptimisedPrunedModel, label = 'Loss')
plt.plot(range(1, epochs + 1), accuracyValuesOfOptimisedPrunedModel, label = 'Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.title('Loss and Accuracy per Epoch')
plt.legend()
plt.show()
```
executed in 296ms, finished 01:43:04 2023-08-11

Figure 21: Plotting Loss and Accuracy per Epoch

References

Nickparvar, M. (2021). Brain tumor mri dataset.
URL: *https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset*

Soewu, T., Singh, D., Rakhra, M., Chakraborty, G. S. and Singh, A. (2022). Convolutional neural networks for mri-based brain tumor classification, *2022 3rd International Conference on Computation, Automation and Knowledge Management (ICCAKM)*, IEEE, pp. 1–7.