UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG



**School of Computer Science & Applied Mathematics**

Brain Tumor Classification Using CNNs

**Course:** Adaptive Computation & Machine Learning

**Submission Date**: 30 May 2025

**Group Members:**

**Names: Student Numbers:**

Lathitha Nongauza 2615978

Fortune Mnisi 1903697

**1.Introduction**

Brain tumors are deadly medical conditions which when treated early can increase a patient's chances of survival. Accurate diagnosis is essential for effective treatment. MRI scans play a key role in detecting and identifying these types of tumors, but interpreting the resultant images can be challenging, even for medical experts.

For this project, we explore the use of deep learning in identifying these tumors, specifically using Convolutional Neural Networks (CNNs). CNN models can help identify and classify brain tumors from MRI images, the goal for this project is to build a model that could accurately distinguish between glioma, meningioma and pituitary tumors. Rather than using pre-trained models, we designed and trained our own CNN model.

This report details the steps we took from preprocessing and augmenting the dataset to building, training, testing and evaluating the model. The aim is to develop a reliable model to solve real-world medical problems.

**1. Dataset and Preprocessing**

This project focuses on classifying brain tumor MRI images into three categories: glioma, meningioma, and pituitary tumor. Given the clinical importance of accurate diagnosis, minimizing misclassifications is critical, as errors can directly impact treatment decisions.

We used the Brain Cancer MRI dataset available on Kaggle. The dataset includes color MRI scans of varying resolutions. To ensure consistency and facilitate processing, all images were normalized and resized from 512 x 512 to 128 × 128 pixels. Images were also converted to tensors and normalized to a standard range to enhance convergence during training.

**2. Data Augmentation**

To increase the diversity of the training data and improve model generalization, we applied data augmentation techniques. Each image was transformed through:

* Random vertical flipping
* Rotations by 90°, 180°, or 270°

This strategy is inspired by previous studies. For example, [Abdou (2022)] used High Power Field (HPF) images of size 1539×1376 pixels, applying 45° incremental rotations and mirroring to expand their dataset. Similarly, [Badža and Barjaktarović (2020)] augmented brain tumor MRI images through 90° rotations and vertical flipping, increasing their dataset threefold.

In our case, the original and augmented images were combined to form a larger, more robust dataset for training and validation.

**3. Dataset Splitting and Validation Strategy**

To prevent overfitting and ensure robust performance estimation, we used 5-fold cross-validation. The full dataset was divided into five equally sized partitions. In each iteration, four partitions were used for training and one for validation. This rotation ensures every data point is used for validation exactly once.

Such a strategy prevents overfitting to a fixed test set and enables the model to be evaluated more comprehensively.

**4. Model Architecture**

Rather than using pre-trained models, we designed a custom Convolutional Neural Network (CNN) suited to our specific dataset. CNNs are ideal for image classification tasks due to their ability to learn spatial hierarchies through convolutional filters.

Our model architecture includes:

* Two convolutional layers with ReLU activations and max-pooling
* A fully connected layer with dropout to prevent overfitting
* An output layer with three units (one per class)

We intentionally started with a simple architecture to reduce computational complexity and better control overfitting.

**5. Training Strategy and Optimisation**

We experimented with various training strategies to address early signs of overfitting. Initially, the training loss converged quickly while the validation loss plateaued at a much higher level, indicating poor generalization.

To address this, we implemented:

* Early stopping, which terminates training when the validation loss increases for three consecutive epochs
* Reduction of input image size to speed up training and encourage better generalization
* Switching optimisers from SGD to Adam, which significantly improved convergence and reduced validation loss

Through experimentation with model complexity, batch sizes, and optimizers, we developed a pipeline that performs well across different validation folds. We trained the model using the 5 batches in parallel. See appendix for the relevant code.

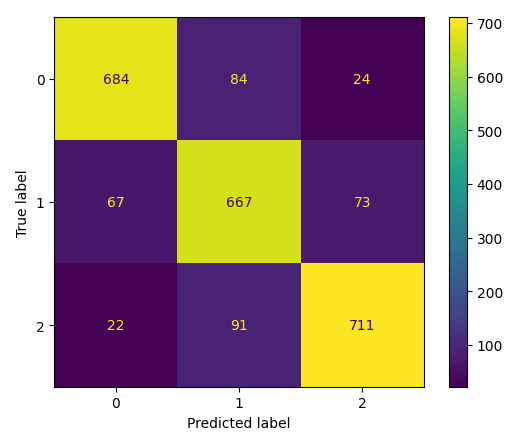
**6. Training results**

| Epoch Loass | Validation Loss |
| --- | --- |
| 0.733054901224105 | 0.5657029645364793 |
| 0.5679288045369058 | 0.541868510793467 |
| 0.5351611318166364 | 0.5214183223052103 |
| 0.4909891165455673 | 0.5313154888934777 |
| 0.45649089176713686 | 0.4888518672497546 |
| 0.40469194752443965 | 0.557837153555917 |
| 0.3541511165384402 | 0.5038998302866201 |
| 0.31113958996509816 | 0.4656613403167881 |
| 0.2602191987627573 | 0.38770941486124133 |
| 0.22920247231736596 | 0.4039165480215041 |



Accuracy on validation set


**7. Test results**

****

Test accuracy: 85.10%

**Summary**

This work demonstrates a complete pipeline for medical image classification using a custom CNN model. Through rigorous preprocessing, data augmentation, K-fold validation, and adaptive training techniques, we achieved high accuracy on brain tumor MRI classification — all while mitigating overfitting and maximizing the reliability of our model’s performance on unseen data.

**Appendix**

Parallel training

**import numpy as np**

**from PIL import Image**

**import torch**

**import random**

**import queue**

**import torch.nn as nn**

**import torch.nn.functional as f**

**from torchvision.transforms import functional as F**

**import torch.optim as optim**

**import torchvision**

**import torchvision.transforms as transforms**

**from torch.utils.data import random\_split**

**from sklearn.model\_selection import KFold**

**from sklearn import metrics**

**import matplotlib.pyplot as plt**

**import multiprocessing**

**class DataHandler:**

**@staticmethod**

**def rotate\_image(x):**

**return F.rotate(x, random.choice([90, 180, 270]))**

**@staticmethod**

**def transform(augment=False):**

**base\_transforms = [**

**transforms.Resize((128, 128)),**

**transforms.ToTensor(),**

**transforms.Normalize([0.5],[0.5])**

**]**

**if augment:**

**augmentation\_transforms = [**

**transforms.RandomVerticalFlip(),**

**transforms.Lambda(DataHandler.rotate\_image)**

**]**

**return transforms.Compose(augmentation\_transforms + base\_transforms)**

**return transforms.Compose(base\_transforms)**

**@staticmethod**

**def dataSplit(trainSet):**

**kf = KFold(n\_splits=5, shuffle=True, random\_state=42)**

**indices = np.arange(len(trainSet))**

**train\_subsets = []**

**val\_subsets = []**

**for train\_idx, val\_idx in kf.split(indices):**

**train\_subsets.append(torch.utils.data.Subset(trainSet, train\_idx))**

**val\_subsets.append(torch.utils.data.Subset(trainSet, val\_idx))**

**return train\_subsets, val\_subsets *# Returns two lists of dataset subsets***

**class Model(nn.Module):**

**def \_\_init\_\_(self):**

**super().\_\_init\_\_()**

**self.conv1 = nn.Conv2d(3, 8, 3, padding=1) *# (8, 128, 128)***

**self.pool = nn.MaxPool2d(2, 2) *# (8, 64, 64)***

**self.conv2 = nn.Conv2d(8, 16, 3, padding=1) *# (16, 64, 64)***

**self.fc1 = nn.Linear(16 \* 32 \* 32, 64)**

**self.dropout = nn.Dropout(0.5)**

**self.fc2 = nn.Linear(64, 3)**

**def forward(self, x):**

**x = self.pool(f.relu(self.conv1(x)))**

**x = self.pool(f.relu(self.conv2(x)))**

**x = torch.flatten(x, 1)**

**x = f.relu(self.fc1(x))**

**x = self.dropout(x)**

**x = self.fc2(x)**

**return x**

**class ModelTest:**

**@staticmethod**

**def plotAccuracy(lossList, label, id):**

**plt.plot(lossList, label=label)**

**plt.xlabel('Episodes')**

**plt.ylabel('Trainig loss')**

**plt.title(f'Performance {id}')**

**plt.legend()**

**plt.show()**

**@staticmethod**

**def plotLoss(lossList, valList,id):**

**plt.plot(lossList, label='Loss')**

**plt.plot(valList, label='Validation')**

**plt.xlabel('Episodes')**

**plt.ylabel('Loss')**

**plt.title(f'Performance {id}')**

**plt.legend()**

**plt.show()**

**@staticmethod**

**def cofusionMatrix(actual, predicted):**

**cm = metrics.confusion\_matrix(actual, predicted)**

**cm\_display = metrics.ConfusionMatrixDisplay(confusion\_matrix = cm, display\_labels = [0, 1, 2])**

**cm\_display.plot()**

**plt.show()**

**class Process(multiprocessing.Process):**

**def \_\_init\_\_(self, train, val, batch, lr, param, id, result\_queue, epochs):**

**super(Process,self).\_\_init\_\_()**

**self.id = id**

**self.train = train**

**self.val = val**

**self.batch = batch**

**self.lr = lr**

**self.param = param**

**self.result\_queue = result\_queue**

**self.epochs = epochs**

**def run(self):**

**trainLoader = torch.utils.data.DataLoader(self.train, batch\_size=32, shuffle=True)**

**validationLoader = torch.utils.data.DataLoader(self.val, batch\_size=32, shuffle=True)**

***#model***

**net = Model()**

***# train***

**numOfEpochs = self.epochs**

**loss = nn.CrossEntropyLoss()**

**optimizer = optim.Adam(net.parameters(), lr=self.lr)**

**totalLoss = []**

**validationLoss = []**

**accuracy = []**

**for epoch in range(numOfEpochs):**

**runningLoss = 0.0**

***#trainig phase***

**for i, data in enumerate(trainLoader):**

**inputs, labels = data**

**optimizer.zero\_grad()**

**outputs = net(inputs)**

**lossValue = loss(outputs, labels)**

**lossValue.backward()**

**optimizer.step()**

**runningLoss += lossValue.item()**

**totalLoss.append(runningLoss / len(trainLoader))**

***# print(f"Process {self.id} Epoch {epoch+1} Loss: {runningLoss/len(trainLoader)}")***

***#validation phase***

**net.eval()**

**correct = 0**

**total = 0**

**with torch.no\_grad():**

**runningLoss = 0.0**

**for i, data in enumerate(validationLoader):**

**inputs, labels = data**

**outputs = net(inputs)**

**lossValue = loss(outputs, labels)**

**runningLoss += lossValue.item()**

**\_, predicted = torch.max(outputs.data, 1)**

**total += labels.size(0)**

**correct += (predicted == labels).sum().item()**

**validationLoss.append(runningLoss / len(validationLoader))**

**accuracy.append(100 \* correct / total)**

***# print(f"Process {self.id} Validation Loss: {runningLoss/len(validationLoader)}")***

**torch.save(net.state\_dict(), self.param)**

**self.result\_queue.put({**

**'id': self.id,**

**'train\_loss': totalLoss,**

**'val\_loss': validationLoss,**

**'accuracy': accuracy**

**})**

**if \_\_name\_\_ == "\_\_main\_\_":**

**data\_dir = "/kaggle/input/brain-cancer-mri-dataset/Brain\_Cancer raw MRI data/Brain\_Cancer"**

**originalData = torchvision.datasets.ImageFolder(root=data\_dir, transform=DataHandler.transform())**

**augmentedData = torchvision.datasets.ImageFolder(root=data\_dir, transform=DataHandler.transform(augment=True))**

***# data***

**allData = torch.utils.data.ConcatDataset([originalData, augmentedData])**

**train\_ratio = 0.8**

**test\_ratio = 0.2**

**train\_size = int(train\_ratio \* len(allData))**

**test\_size = len(allData) - train\_size**

**train\_set, test\_set = random\_split(allData, [train\_size, test\_size])**

**train\_batches, val\_sets = DataHandler.dataSplit(train\_set)**

***# parallel training***

**result\_q = multiprocessing.Queue()**

**results = []**

**processes = [**

**Process(train\_batches[0], val\_sets[0], 0, 0.0008, 'model1.pth', 1, result\_q, 20),**

**Process(train\_batches[1], val\_sets[1], 1, 0.0001, 'model2.pth', 2, result\_q, 20),**

**Process(train\_batches[2], val\_sets[2], 2, 0.0005, 'model3.pth', 3, result\_q, 20),**

**Process(train\_batches[3], val\_sets[3], 3, 0.005, 'model4.pth', 4, result\_q, 20),**

**Process(train\_batches[4], val\_sets[4], 4, 0.001, 'model5.pth', 5, result\_q, 20)**

**]**

**try:**

**for p in processes:**

**p.start()**

**for p in processes:**

**p.join(timeout=86400) *# 24-hour timeout***

**results = []**

**while len(results) < len(processes):**

**try:**

**results.append(result\_q.get(timeout=14400))**

**except queue.Empty:**

**print("Timeout waiting for results!")**

**break**

***# Plot results***

**for res in sorted(results, key=lambda x: x['id']): *# Plot in order***

**ModelTest.plotLoss(res['train\_loss'], res['val\_loss'], res['id'])**

**ModelTest.plotAccuracy(res['accuracy'], "Accuracy",res['id'])**

**except Exception as e:**

**print(f"Error: {e}")**

**for p in processes:**

**if p.is\_alive():**

**p.terminate()**

**finally:**

**for p in processes:**

**p.join()**

***# model test***