Deep Learning Approaches for Brain Tumor Classification

Arda Öztüner, 99735831140

Department of Computer Engineering, ardaoztuner@ogr.eskisehir.edu.tr

***Abstract*—Brain tumor classification has a very important place in the medical field. Determining the tumor in the earliest and most accurate way in the diagnosis section can positively advance the treatment. In this project, different deep learning approaches for brain tumor classification were examined. A total of five deep learning models were trained, including two different special CNN models and pre-trained models based on ResNet50, MobileNetV2 and DenseNet121. The dataset was processed with data augmentation, normalization and histogram equalization processes before training. The performances of the trained models were measured and compared with metrics such as Accuracy, Precision, Recall and F1-Score. The obtained results showed the power of deep learning in medical applications.**

***Index Terms*—Machine Learning, Deep Learning, Brain Tumor Classification, Convolutional Neural Networks, Medical Imaging, Project Report, Transfer Learning**

I. Introduction

Timing and correct diagnosis play a critical role for patients when detecting brain tumors. Incorrect or delayed diagnoses can lead to serious consequences. In traditional methods, doctors can take a long time to manually examine and interpret X-ray images. This is where artificial intelligence shows itself at today's point. This study particularly demonstrates how effective deep learning algorithms are in medical imaging. The aim of this project is to classify brain tumors from X-ray images into 2 classes as "Healthy" or "Tumor" using both special and pre-trained deep learning algorithms. Examples of these images are shown in Figure 1, providing a visual representation of the dataset.

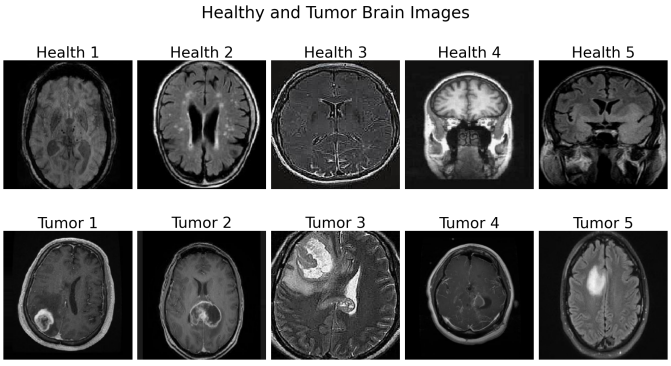

Fig. 1. Example images from the dataset.

The dataset used in this study was carefully analyzed to ensure balance between classes.

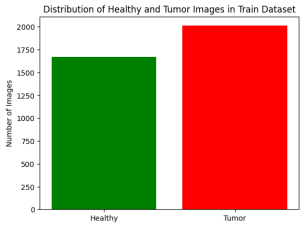

Fig. 2. Distribution of Training Data

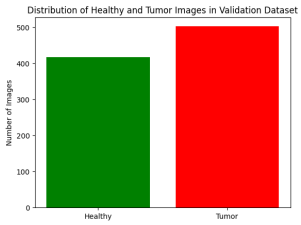

Fig. 3. Distribution of Validation Data

Figure 2 illustrates the distribution of training data, while Figure 3 depicts the distribution of validation data. Maintaining such balance is critical for avoiding biased model predictions and achieving robust classification results. As can be seen from the graphs, there is no imbalance in our data set, we just have more tumor data, but this small amount of excess does not pose a problem. A total of 5 models were created, 2 of which are specially designed CNN architectures and the other 3 are pre-trained architectures (ResNet50, MobileNetV2, and DenseNet121).

II. Proposed Methodology

A. *Model Selection*

In this project, 5 deep learning models were trained to classify brain images as "Healthy" and "Tumor". These models included two custom-designed convolutional neural networks (CNNs) and three pre-trained architectures: ResNet50, MobileNetV2, and DenseNet121.

Before training the models, several image processing techniques were applied to the dataset to enhance its quality and variability. Figure 4 illustrates an example image from the dataset after applying these processing steps.

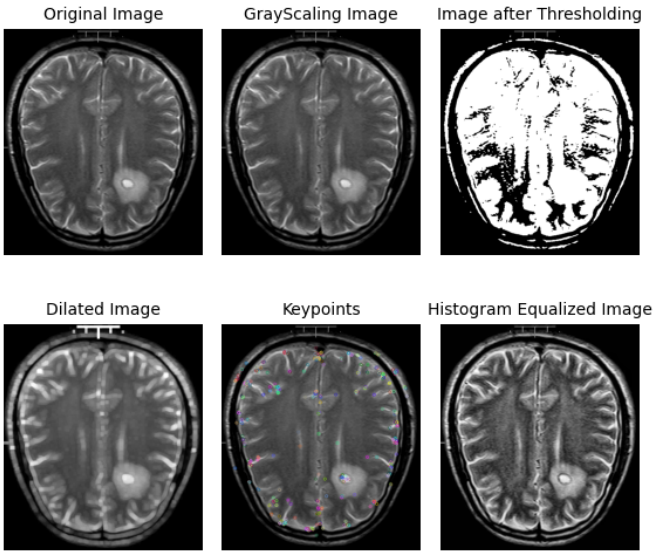

Fig. 4. Example images after image processing techniques.

Figure 5 shows the pixel intensity histogram for the "Healthy" class before any processing was applied, while Figure 6 illustrates the pixel intensity histogram for the "Tumor" class before histogram equalization.

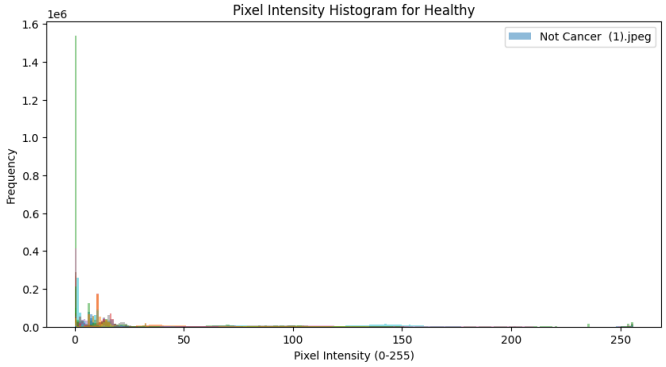

Fig. 5. Pixel intensity histogram for "Healthy" class before histogram equalization.

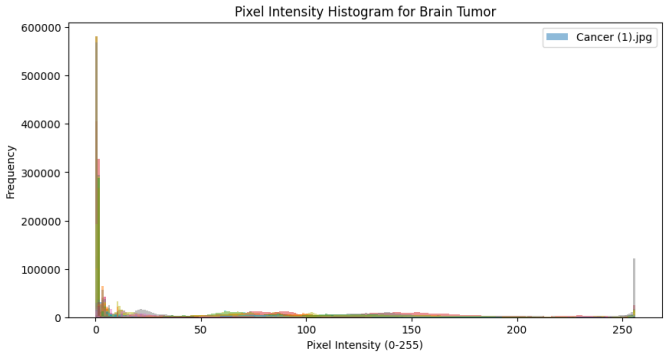

Fig. 6. Pixel intensity histogram for "Tumor" class before histogram equalization.

As a foundational model, the first customized CNN used a simple structure consisting of three convolutional layers followed by max-pooling and fully connected layers. In order to improve training stability and generalization, the second custom CNN added Batch Normalization and Dropout layers to this design. For the pre-trained models, ResNet50 was chosen for its residual connections, which allow for efficient gradient flow and make it highly effective for complex image classification tasks. MobileNetV2 was chosen because of its computationally efficient and lightweight design, which makes it perfect for applications that need low latency. Lastly, DenseNet121 was included due to its unique dense connections that encourage feature reuse and optimize parameter efficiency. These pre-trained models were improved on the brain tumor classification dataset by leveraging transfer learning, which enabled the models to exploit information from large-scale datasets.

B. Implementation Strategy

For the custom CNNs, TensorFlow and Keras were used to design deep learning models with convolutional, max-pooling, and dense layers. While the first CNN used a simple structure to set baseline performance, the second custom CNN added Batch Normalization for quicker and more stable convergence and Dropout layers to reduce overfitting. Binary cross-entropy loss and the Adam optimizer were used to construct both models, and "accuracy" was used as the evaluation metric. Transfer learning was used to modify the pre-trained models, ResNet50, MobileNetV2, and DenseNet121. To preserve the previously obtained features from ImageNet, the basic layers of these models were first frozen. To improve the models for the binary classification, additional layers were placed on top, including fully linked layers with Batch Normalization and Dropout. After initial training, all layers of ResNet50 were unfrozen to enable fine-tuning of the whole model, maximizing performance for the particular dataset. Due to its lightweight nature, MobileNetV2 kept its foundation fixed and concentrated on retraining the top layers in order to maximize its efficiency while preserving computational viability. The addition custom classification layers allowed DenseNet121 to take use of its dense connectivity.A batch size of 32 was used to train the models, and early stopping was used to avoid overfitting, ensuring effective and reliable training.

C. Evaluation Metrics

A set of evaluation metrics was used to evaluate the models' performance. The primary metric of the model's performance was accuracy, which calculates the percentage of properly identified instances among all predictions. However, other measures including accuracy, recall, and F1-score were also used to provide a deeper evaluation because of the possibility of class imbalance in medical datasets.

D. Testing Setup

Google Colab, a cloud-based platform that offers access to high-performance computing resources, served as the site for the project's trainings. The free GPU support offered by

Google Colab was used to ensure effective processing and speed up model training. NVIDIA Tesla T4 GPUs, which are appropriate for deep learning model training on huge datasets, were the main piece of hardware utilized in the system. The development interface was Jupyter Notebook in Colab, which provided an engaging and adaptable coding environment.

III. Experiments and Results

A. *Basic CNN 1*

The model is first trained using a simple CNN architecture with three convolutional layers, followed by a max pooling layer and a fully connected layer.The model layers can be seen in Figure 7.

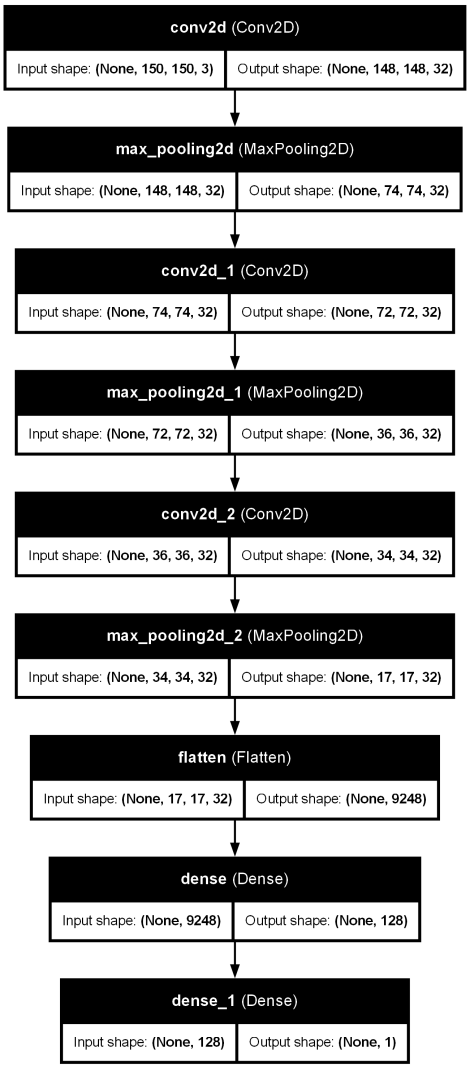

Fig. 7. Model architecture of First CNN

Training and Validation Loss/Accuracy Graphs: Figure 8 shows both the training and validation loss curves as well as the training and validation accuracy. These graphs provide insights into the model's learning behavior, including overfitting and convergence. According to the graphs, it is seen that there is no overfitting and the learning occurs stably.

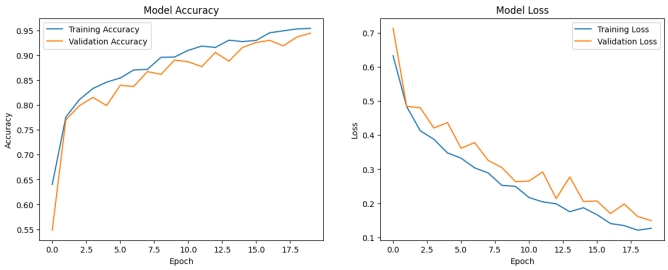

Fig. 8. Training and Validation Loss and Accuracy for CNN 1

Confusion Matrix: Figure 9 shows the confusion matrix for first CNN, which highlights the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). This matrix helps evaluate how well the model performed in classifying both "Healthy" and "Tumor" images.

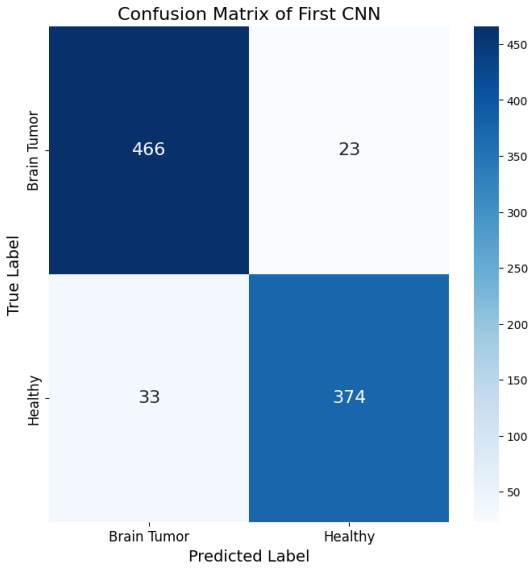

Fig. 9. Confusion Matrix for Basic CNN 1

B. *Basic CNN 2*

The second model, Basic CNN 2, is an enhanced version of Basic CNN 1, with added Batch Normalization and Dropout layers to improve stability and generalization.The model layers can be seen in Figure 10.

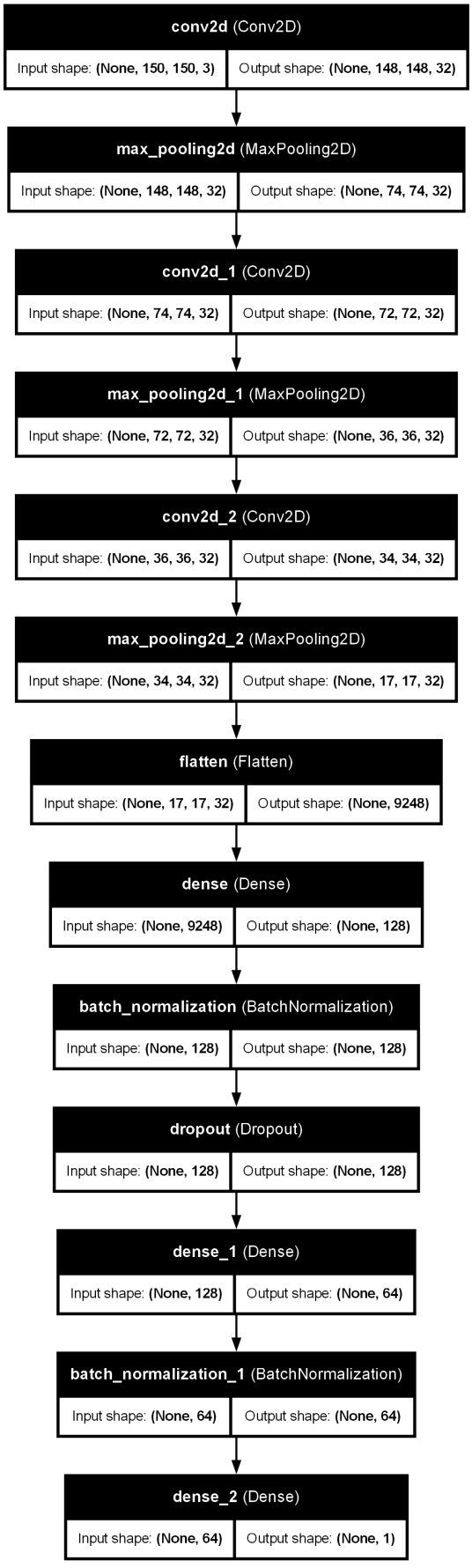

Fig. 10. Model architecture of Second CNN

Training and Validation Loss/Accuracy Graphs: Figure 11 shows both the training and validation loss curves as well as the training and validation accuracy. From the graphs here, it can actually be said that this model exhibits a more irregular structure than the previous CNN model. Since it is a more complex structure, Validation Accuracy has experienced constant ups and downs, making it a bit difficult to learn with this model. We can understand from here that the first model, which is simpler, has shown better success than this model.

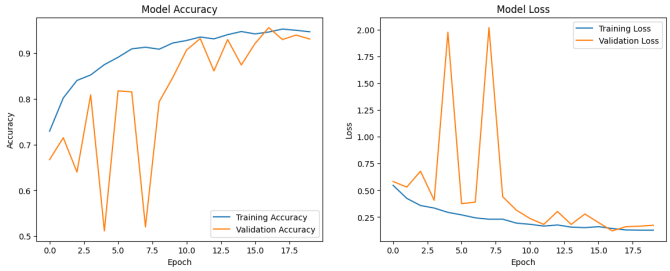

Fig. 11. Training and Validation Loss and Accuracy for CNN 2

Confusion Matrix: Figure 12 shows the confusion matrix for second CNN, The findings of the two confusion matrices obtained from both CNNs show that, despite using advanced techniques like Batch Normalization and Dropout, the second CNN performed worse than the first CNN. This poor performance could be due to a variety of circumstances. The combination of Batch Normalization and Dropout may have resulted in over regularization. Over-regularization can impair the model's capacity to understand subtle patterns in the data, leading to a decrease in accuracy.

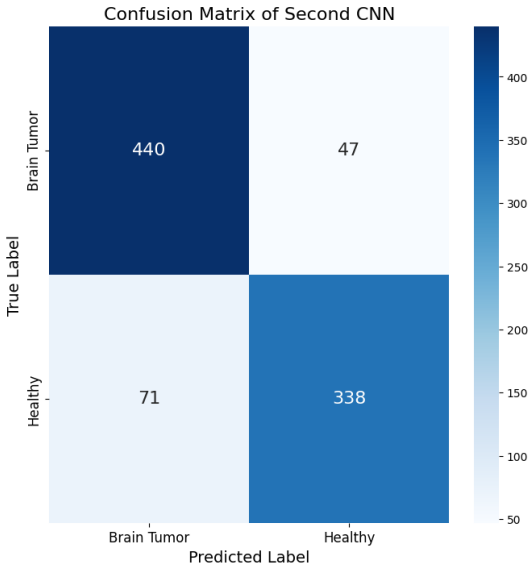

Fig. 12. Confusion Matrix for Basic CNN 2

C. *ResNet50 Model*

The ResNet model was developed with the ResNet50 architecture, which is a deep convolutional neural network that uses residual learning to solve the vanishing gradient

problem. This ResNet50 architecture was employed using transfer learning for image classification. Transfer learning leverages pre-trained models, in this case, the ResNet50 model pre-trained on the ImageNet dataset, and fine-tunes it to the specific task of brain tumor classification.

The base model, ResNet50, was initialized with ImageNet weights, and the top layer was removed to adapt the model to our classification objective. Then built unique layers on top of the pre-trained base to best address the situation at hand. The following custom layers were utilized:

- **Flatten**
- **Dense (512 units, He initialization)**
- **Dropout (0.4)**
- **BatchNormalization**
- **Activation (ReLU)**
- **Dense (128, 16 units)**
- **Output Sigmoid Layer**

Figure 13 shows the training and validation loss curves, as well as the accuracy metrics, for the ResNet model.

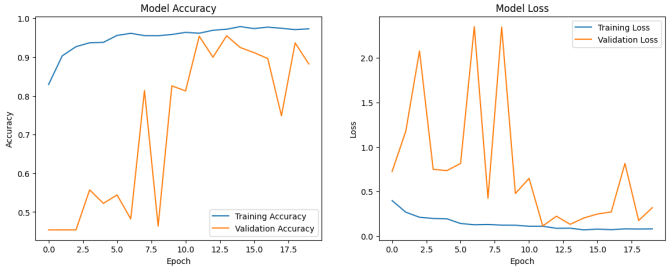

Fig. 13. Training and Validation Loss and Accuracy for ResNet Based Model

Confusion Matrix: Figure 14 shows the confusion matrix for ResNet50 based model.

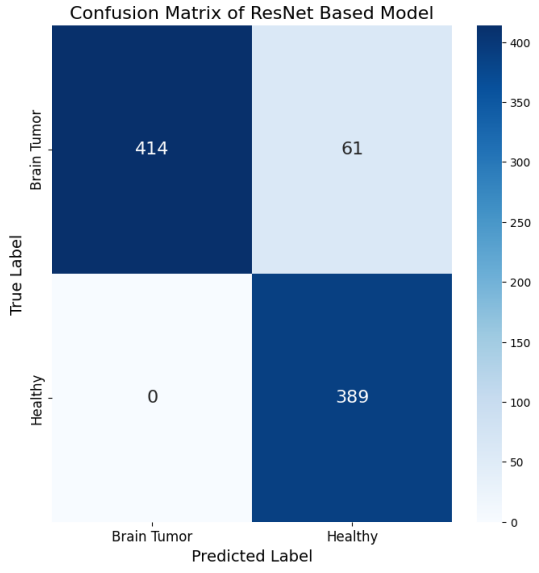

Fig. 14. Confusion Matrix for ResNet Based Model

D. *MobileNetV2 Model*

MobileNetV2 is an efficient deep learning model that is optimized for mobile and edge devices, providing a good combination of performance and computational cost. MobileNetV2 benefits from transfer learning by employing ImageNet's pre-trained weights, allowing for faster convergence and improved performance even with fewer datasets.

The MobileNetV2 base model is initialized with pre-trained weights from ImageNet, excluding the top classification layer. The top layer was then replaced with custom layers suitable for binary classification. The following custom layers were added:

- **Flatten**
- **Dense (512 units, ReLU activation)**
- **BatchNormalization**
- **Dropout (0.4)**
- **Output Sigmoid Layer**

Figure 15 shows the training and validation loss curves, as well as the accuracy metrics, for the MobilNetV2 based model.

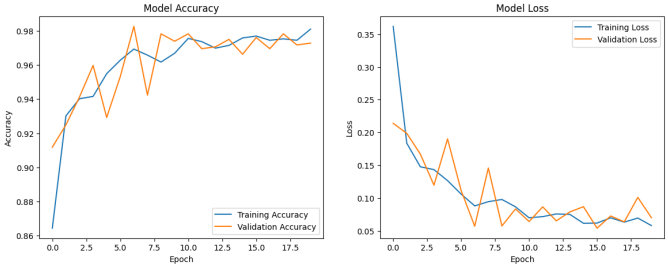

Fig. 15. Training and Validation Loss and Accuracy for MobilNetV2 Based Model

Confusion Matrix: Figure 16 shows the confusion matrix for MobilNetV2 based model.

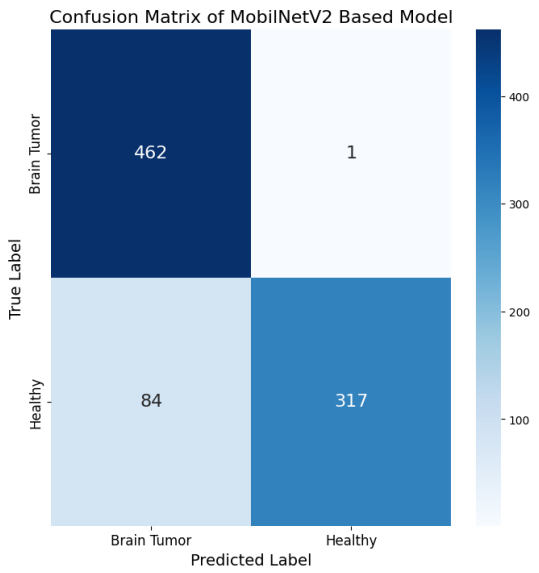

Fig. 16. Confusion Matrix for ResNet Based Model

E. *DenseNet121 Model*

DenseNet121 is a densely connected convolutional network that benefits from direct connections between each layer to every other layer in the network. This design enhances feature reuse, leading to more efficient learning, especially for complex image classification tasks. The model is initialized with pre-trained weights from ImageNet, with the top layer removed and replaced with custom layers for binary classification. The custom layers added are as follows:

- **Flatten Layer**
- **Dropout (0.7)**
- **BatchNormalization**
- **Dense (16 units, ReLU activation)**
- **Dropout (0.5)**
- **Output Layer (Sigmoid activation)**

Figure 17 shows the training and validation loss curves, as well as the accuracy metrics, for the DenseNet121 based model.

Fig. 17. Training and Validation Loss and Accuracy for DenseNet121 Based Model

Confusion Matrix: Figure 18 shows the confusion matrix for DenseNet121 based model.

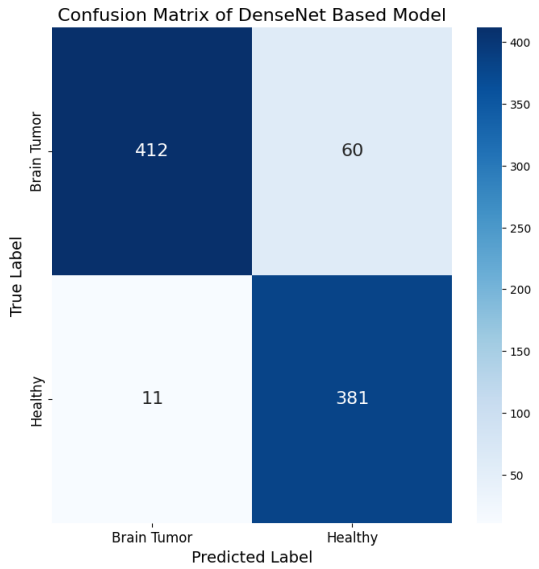

Fig. 18. Confusion Matrix for DenseNet121 Based Model

F. *Conclusion*

The performance of all trained models on the validation dataset can be seen in the Table I.

Model	Accuracy	Precision	Recall	F1 Score
First CNN	95%	95%	95%	95%
Second CNN	87%	87%	86%	87%
ResNet50	92%	93%	93%	93%
MobileNetV2	91%	93%	91%	92%
DenseNet121	92%	93%	91%	92%

TABLE I
COMPARISON OF MODEL PERFORMANCE METRICS

IV. CODE EXPLANATION

The project is well-structured and organized into multiple directories and files, each serving a specific purpose. Below is a detailed explanation of the code structure:

A. *Project Structure*

The project follows a modular and logical organization, ensuring that each component of the workflow is encapsulated. The directory structure is as follows:

- **data/**: This directory contains the datasets used in the project.
  - **raw/**: Stores the unprocessed dataset, original X-ray images.
  - **processed/**: Contains the preprocessed dataset after applying transformations like normalization, histogram equalization, and data augmentation.
- **src/**: This is the core directory where all the main functionalities of the project are implemented.
  - **data_loading.py**: Contains functions for loading and preprocessing data, including splitting datasets into training, validation, and test sets.

    1. load_data(data_dir, img_size=(150, 150), batch_size=32): This function loads brain tumor data from the specified directory and applies data augmentation techniques using TensorFlow's `ImageDataGenerator`. It splits the dataset into training (80%) and validation (20%) subsets. Images can be resized to a specific dimension ('img_size'), which defaults to 150x150 pixels, or the original dimensions can be preserved. Data augmentation techniques, such as rescaling pixel values, random shearing, zooming, and horizontal flipping, are applied to improve the model's generalization capability. The `flow_from_directory` method is used to generate TensorFlow-compatible datasets, enabling efficient handling of large datasets in memory. The function returns two generators: one for training and another for validation.

    2. get_random_image(data_dir): This function selects a random image from the specified directory and returns it in RGB format. It first scans the directory for image files with extensions such as `.png`, `.jpg`, and `.jpeg`, and stores their paths in a list. A random image is selected from this list, and the image is loaded using OpenCV. If no images are found in the directory, the function raises a

`ValueError`. This function is particularly useful for exploring the dataset visually or for testing preprocessing techniques.

– **model.py**: This module contains functions for creating different deep learning models used in the project. Each function is responsible for defining and compiling a specific architecture. Below is a brief overview of the functions:
  * **create_basic_cnn1**: Builds a simple CNN model with three convolutional layers, followed by max pooling and fully connected layers.
  * **create_basic_cnn2**: Constructs an enhanced version of the first CNN with additional Batch Normalization and Dropout layers for improved generalization.
  * **create_resnet_model**: Implements a ResNet50-based architecture using transfer learning, with added custom fully connected layers.
  * **create_mobilenetv2_model**: Defines a MobileNetV2-based architecture, leveraging pre-trained weights from ImageNet.
  * **create_densenet_model**: Creates a DenseNet121-based architecture with dense connections and additional regularization layers.

– **visualization.py**: This module contains functions to visualize images and model performance metrics. These functions allow for effective exploration of the dataset, model evaluation, and performance reporting. Below are the detailed explanations of each function:

1. visualize_image(data, label=None): This function selects a random image from the given dataset and visualizes it. If a specific label (either "Health" or "Tumor") is provided, the function will only display a random image from the specified label. If no label is provided, it selects a random image from the dataset.
2. visualize_images_by_label(data, num_images_per_label=5): This function visualizes random images from both the "Health" and "Tumor" categories in the given dataset. The user can specify how many images to display per label. The function will display the images in two rows, one for "Healthy" images and the other for "Tumor" images. This helps in understanding the distribution of different labels in the dataset visually.
3. visualize_class_distribution(data): This function plots a bar chart showing the distribution of "Healthy" and "Tumor" images in the dataset. This is useful for ensuring balanced class distribution in the training or validation dataset.
4. plot_training_history(history): This function plots the training and validation accuracy and loss curves based on the model's training history. The function generates two subplots: one for accuracy and the other for loss, showing the model's performance over epochs. This helps to monitor model convergence and detect potential overfitting.
5. display_confusionMatrix_and_report(model, validation_generator, step_size_valid, n_classes, class_names, model_name="Model"): This function generates a confusion matrix and classification report for evaluating the performance of a trained model. This is crucial for evaluating model performance.
6. display_image_properties(data_dir): This function displays statistical summaries and histograms of image properties such as height, width, and color channels (e.g., RGB or grayscale). It processes images in the "Brain Tumor" and "Healthy" categories and generates histograms of image dimensions, providing an overview of the image sizes in the dataset.
7. display_pixel_intensity(data_dir): This function generates pixel intensity histograms for images in the "Brain Tumor" and "Healthy" categories. It reads images, computes their pixel intensity distributions, and visualizes them using histograms. This is important for understanding the intensity distribution of images in both categories, which can inform preprocessing or normalization decisions.

– **utils.py**: This module contains a collection of helper functions used for image processing and statistical analysis of pixel values. These functions enable efficient data preparation, image enhancement, and analysis, which are crucial for understanding and preprocessing the dataset. Below are the detailed explanations of each function:
1. convert_to_grayscale(image): This function takes an input image in RGB format and converts it to grayscale.
2. apply_threshold(image, threshold_value=70): This function applies binary thresholding to an input grayscale image, converting it into a binary image.
3. apply_dilation(image, kernel_size=(6, 6), iterations=1): This function applies dilation, a morphological operation, to a binary image. Dilation expands the boundaries of the white areas in the image.
4. detect_and_draw_keypoints(image): This function detects keypoints in the input image using the ORB (Oriented FAST and Rotated BRIEF) feature detector and descriptor extractor.
5. adaptive_histogram_equalization(image, clip_limit=2.0, grid_size=(16, 16)): This function applies adaptive histogram equalization (CLAHE) to enhance the contrast of a grayscale image. CLAHE is particularly useful for improving the contrast of images with varying lighting conditions.
6. apply_histogram_equalization_to_folder(folder_path, output_folder): This function applies adaptive histogram equalization to all images in a specified folder and saves the resulting images to another

folder.

7. pixel_intensity_stats(data_dir): This function calculates and prints statistical summaries (mean, median, standard deviation, minimum, and maximum) of pixel intensities for images in the "Brain Tumor" and "Healthy" categories. It processes all the images in the specified directory, computes the statistics for each category, and prints the results. This function helps in understanding the distribution of pixel intensities across the two categories.

8. analyze_pixel_values(data_dir): This function analyzes the pixel values of images in the "Brain Tumor" and "Healthy" categories to check for normality and perform statistical tests. It uses the Shapiro-Wilk test for normality and, depending on the results, performs either an independent t-test (if both categories are normally distributed) or a Mann-Whitney U test (if they are not). The function prints the results of these statistical tests and determines whether there is a statistically significant difference between the two categories.

- **notebooks/**: This directory contains Jupyter notebooks used for experimentation and analysis.
  - **Exploratory_data_analysis.ipynb**: Performs initial analysis on the dataset, visualizing class distributions and exploring features.
  - **Modelling.ipynb**: Used for developing and testing different deep learning architectures before finalizing them.
  - **Inference.ipynb**: Demonstrates how to load trained models and make predictions on new data.
- **experiments/**: This notebook contains the experimental results and analysis conducted during the development and optimization of the CNN models used for brain tumor classification. The key focus of the experiments was to observe and compare the impact of histogram equalization on model performance.The experiment confirmed that histogram equalization can positively impact the performance of CNN models for brain tumor classification. By enhancing the contrast of the images, histogram equalization enables the models to better distinguish between the "Healthy" and "Tumor" classes. This preprocessing step was applied to the entire dataset to ensure consistency and improve model accuracy, making it an essential technique for improving classification performance in this task.
- **models/**: This directory stores the trained models and their associated training histories for the deep learning models used in this project. This directory includes the final model weights in `.h5` format, which can be reloaded for inference or further fine-tuning, as well as the training histories in `.json` format that record the training progress and performance metrics for each model.
- **results/**: Contains visualizations of results, such as accuracy and loss graphs, confusion matrices, and image examples.
- **requirements.txt**: Specifies all the dependencies required to run the project, ensuring a consistent environment across systems.

V. Personal Contributions

As the sole contributor to this project, I was responsible for all aspects of its development, from data collection and preprocessing to model design, training, and evaluation.

VI. Discussion and Conclusion

Several deep learning models, including pre-trained models like ResNet50, MobileNetV2, and DenseNet121, as well as customized CNN architectures, were trained and evaluated for brain tumor classification in this study. Following extensive testing, it came out that the customized CNN1 model performed better than the others in terms of generalization and accuracy. First CNN showed a more stable training process as well as better performance on histogram-equalized datasets, while having a simpler architecture than more intricate models like ResNet50 and DenseNet121.

CNN1's higher performance might be explained by its simpler design, which, when combined with appropriate regularization strategies, allowed for faster convergence and a lower chance of overfitting. However, despite their ability to extract deep features, advanced models like ResNet50 and DenseNet121 could not have been able to generalize well on this specific dataset because of their larger capacity, which might have led to overfitting or delayed convergence.

The experiments also demonstrated that histogram equalization improves model performance. Models trained on histogram-equalized pictures outperformed models trained on raw images, emphasizing the significance of preprocessing approaches in medical image classification tasks.

In conclusion, while advanced models like as ResNet50 and DenseNet121 produce promising results, the customized CNN1 model was the most successful for this job, proving that simpler architectures may occasionally produce better outcomes when correctly calibrated.

VII. Ethical Statement

I confirm that all experimental work in this project was carried out by me with integrity. All sources and methods used in the study have been properly cited and included in the references. No plagiarism has been committed in any part of this study.

VIII. AI Tools Usage

In this project, AI tools were utilized for efficiently managing image files and directories, specifically using the *os* library in Python. The *os* library was essential for reading and organizing the image dataset, as it allowed for automated navigation of directories, loading images from various folders, and handling file paths. AI tools were used to speed up and make operations with the *os* library more efficient.

IX. References

References

[1] ScienceDirect. (2023). *Introduction to deep learning and diagnosis in medicine. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/B9780323961295000032*

[2] ScienceDirect. (2022). *A survey of modern deep learning based object detection models. Retrieved from https://www.sciencedirect.com/science/article/pii/S1051200422001312*

[3] Pierre, L. (2024). *Transfer Learning with DenseNet-121*. Medium. Retrieved from https://medium.com/@pierre.lgsm/transfer-learning-with-densenet-121-1be8afbb568d

[4] PMC. (2021). *Convolutional neural networks in medical image understanding*. Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC7778711/

[5] MDPI. (2024). *Medical Image Classifications Using Convolutional Neural Networks*. Retrieved from https://www.mdpi.com/2504-4990/6/1/33

[6] ScienceDirect. (2021). *A scoping review of transfer learning research on medical image analysis using ImageNet*. Retrieved from https://www.sciencedirect.com/science/article/pii/S0010482520304467

[7] Towards Data Science. (2017). *Histogram Equalization: A Comprehensive Guide*. Retrieved from https://towardsdatascience.com/histogram-equalization-5d1013626e64

[8] P. Viradiya, "Brain Tumor Dataset," 2020. [Online]. Available: https://kaggle.com/datasets/preetviradiya/brian-tumor-dataset/data?select=Brain+Tumor+Data+Set.