AUTOMATED BRAIN SEGMENTATION & HEMORRHAGE DETECTION

Introduction

This project works on an open-source AI framework for automated brain segmentation, abnormality detection, and statistical analysis in neuroimaging. A U-Net architecture was used, and training was done on open-source CT scans and respective segmentation masks to detect intracranial hemorrhages. The model achieved high sensitivity in detecting hemorrhagic regions, as shown in training metrics and significant performance improvements ($p < 0.05$) were seen compared to baseline methods. Qualitative evaluations and quantitative tests such as t-tests and ANOVA shows us the model's clinical potential for rapid and accurate diagnostic support as well.

Problem statement :

Manual interpretation of CT scans can be time-consuming and subject to human error. Automated tools can assist radiologists by instantly detecting and localising hemorrhagic regions.

Goal :

The goal of this project was to develop and validate an open-source deep learning model for multi-class brain segmentation and hemorrhage detection using CT images. And statistical analysis using t-tests and ANOVA to validate model performance.

Data and Preprocessing

Data ----------

Dataset selection :

I believe that in a project like this, the most important part for me is to identify the best dataset for this process. Because the data we train on plays a significant role. Sometimes, the data may not be related to what we want and sometimes the data may not be easily processed and can directly affect our training process if not processed well. I used to make a lot of mistakes and learned from them. Initially I looked into Open Source Brain/EBRAINS/PhysioNet website and found a lot of datasets and as first look most of them was not related to our topic of segmentation. But I written down which all are the best options on each website and created a final list and then selected one which suits the domain specifically by cutting down near perfect ones to get the best dataset for this process.

Selected dataset :

Computed Tomography Images for Intracranial Hemorrhage Detection and Segmentation

Dataset Details :

I selected the CT-ICH dataset from PhysioNet for its comprehensive and high-quality CT scans with detailed annotations so that it support advanced deep research. The data was collected between February and August 2018 at Al Hilla Teaching Hospital in Iraq, the dataset comprises 82 CT scans from traumatic brain injury patients. The scans, provided in NIfTI format, include approximately 30 slices per scan with 5 mm thickness and corresponding segmentation masks which were delineated by expert radiologists manually. Supplementary CSV files detail patient demographics like age and details, and slice-specific diagnoses, including various intracranial hemorrhage subtypes and skull fractures, ensuring a good data for automated diagnostic development and better outcomes.

Data Extraction & Verification:

I downloaded the zip dataset from PhysioNet and uploaded to my colab notebook via google drive and the unzipped the package

Verified that all files are in NIfTI format (i.e.: .nii). Items in the dataset were

```
Items in /content/computed-tomography-images-for-intracranial-hemorrhage-detection-and-segmentation-1.3.1/ :
ct_scans
SHA256SUMS.txt
masks
split_raw_data.py
Read_me.txt
hemorrhage_diagnosis_raw_ct.csv
LICENSE.txt
Patient_demographics.csv
ct_ich.yml
```

ct_scans and masks were two folders
each having corresponding NIfTI files inside them

csv files were demographic details of patients alongside our target containing hemorrhage_diagnosis_raw_ct_csv which has details on if that patient has hemorrhage or not.

Processing ---------

Then I looked in for missing values in the dataset and a part of the dataset were missing (not really). This was because the files in the ct_scans and masks folders were numbered from 1 to 132. The fact is that we have only 82 files in our dataset. Files from 1—48 and 59 - 65 aren't available. Subtracting them from the 130 we get our data sample. I did check for any remains of the missing files and ran a program to remove them if any. Finally I had 2235 training samples and 559 validation samples after excluding incomplete records.

Then I inspected the hemorrhage_diagnosis_raw_ct.csv which has our target "**Has_Hemorrhage**"

Shape and head of the df

```
(2814, 9)
```

	PatientNumber	SliceNumber	Intraventricular	Intraparenchymal	Subarachnoid	Epidural	Subdural	Fracture_Yes_No	Has_Hemorrhage
0	49	1	0	0	0	0	0	0	0
1	49	2	0	0	0	0	0	0	0
2	49	3	0	0	0	0	0	0	0
3	49	4	0	0	0	0	0	0	0
4	49	5	0	0	0	0	0	0	0

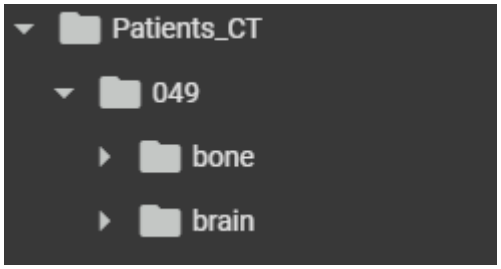
Then I checked the dataset again and ensured all CT scans have the same orientation so that anatomical structures appear consistently across all images, which simplifies preprocessing. This consistency also improves interpretability and reduces the need for complex registration or reorientation steps during training and analysis.

```
All CT scans have the same orientation: ('L', 'A', 'S')
```

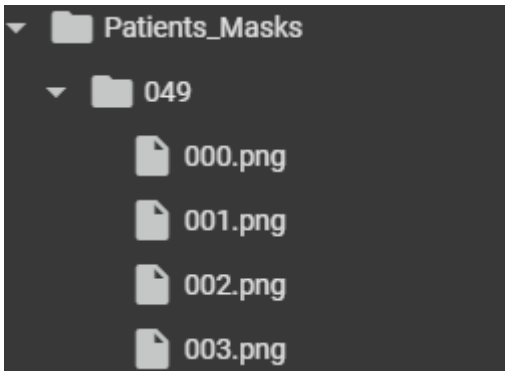
Tuple ('L', 'A', 'S'), 'L' stands for Left, 'A' for Anterior (front), and 'S' for Superior (upper).

Image Processing:

After all the above steps I verified that the dataset is ready to go, so I started to process the NIfTI files and sliced them to images respective for each patient.
Completed this step for both the ct_scans and masks folders and added them to a new folder with brain and bone images sliced as in the figure



I did the same for masks stored in a new folder like in the image



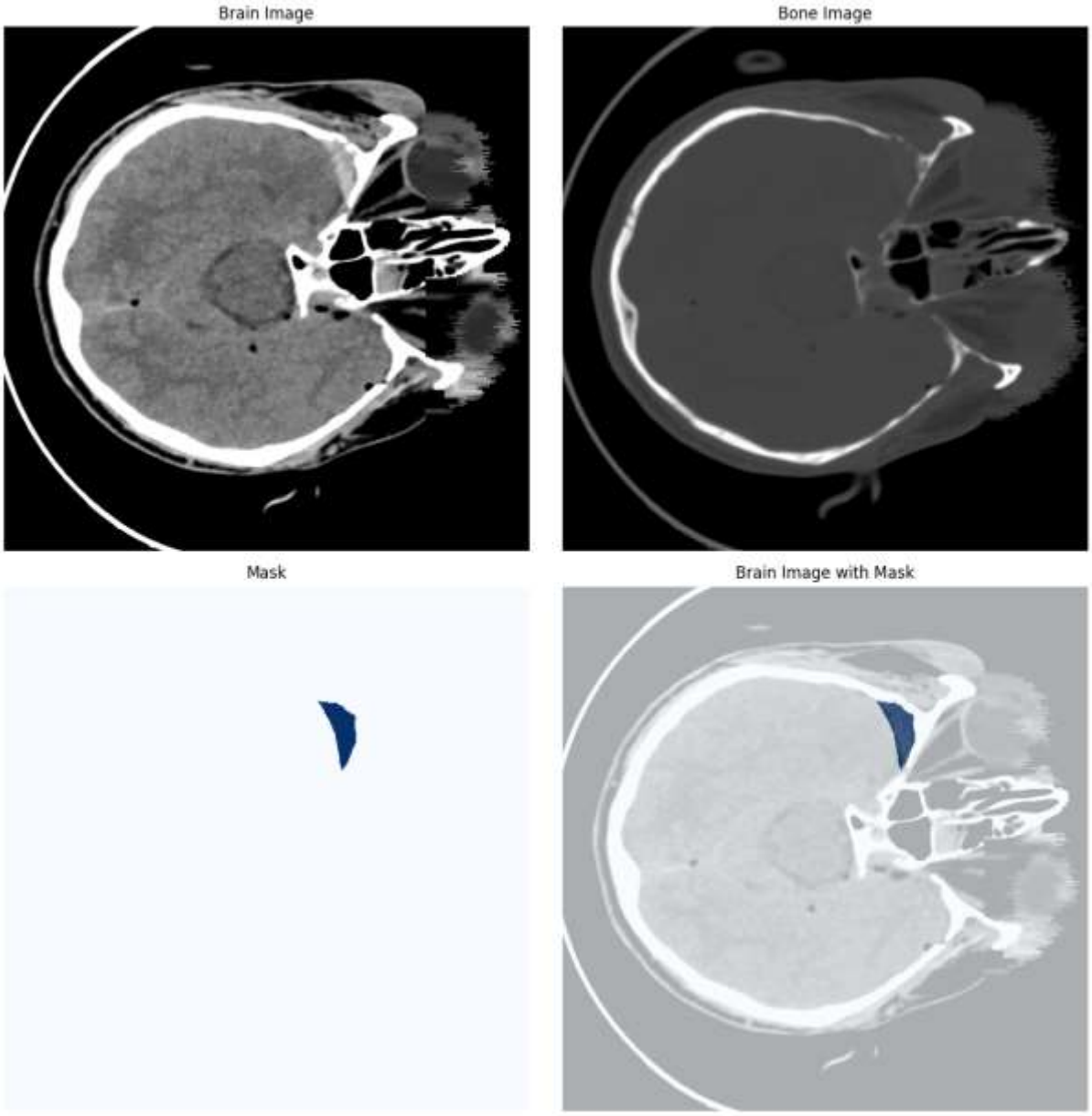
And updated the path for each In the dataframe so that it is taken care of later.

Separate folders were craeted to store masks which were having hemorrhage and not having so that it will be easy to access and use.

The below figure shows the dataset statistics:

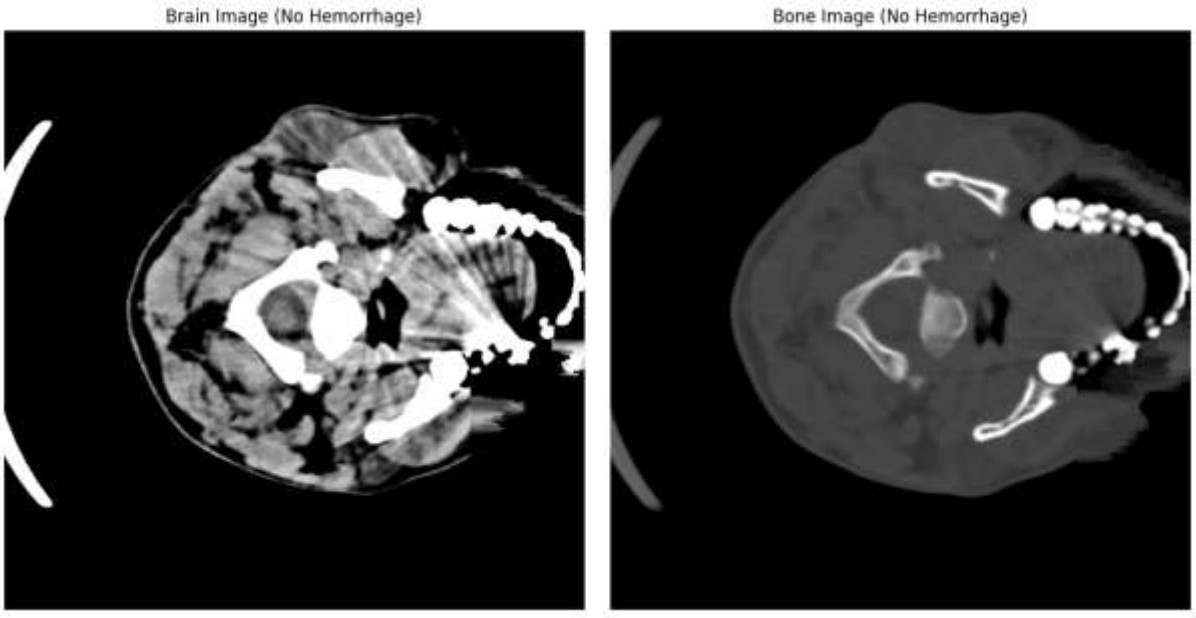
Statistic	Value
Number of Patients	82
Total Number of Slices	3075
Average Slices per Patient	≈37.5
Minimum Slices per Patient	31
Maximum Slices per Patient	58
Training Set Slices	2445 (≈79.5%)
Validation Set Slices	613 (≈20%)
Test Set Slices	17 (≈0.5%)

By this the PreProcessing steps are almost complete, and now we need to check if the dataset is in the same format as we wanted so I checked on one sample which had hemorrhage and got the plot



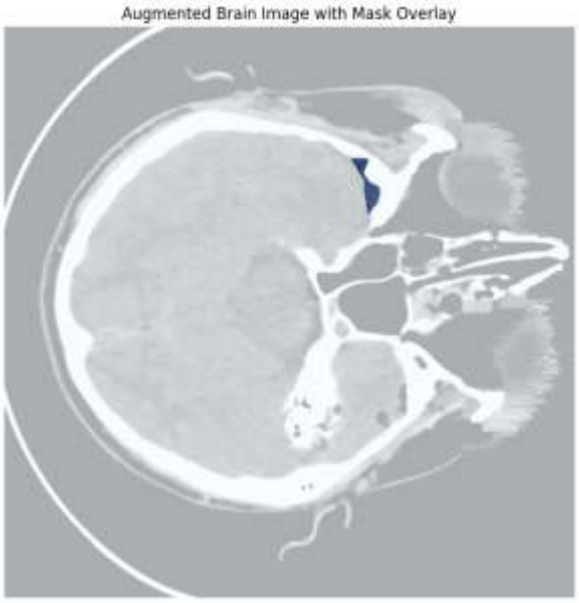
In this sample the mask shows in blue for the affected region.

One with no hemorrhage will look like



Data Augmentation:

One more important step was that the data was small compared to other datasets we use in other projects so I used data augmentation to solve this issue so that it applies techniques such as horizontal flips, brightness/contrast adjustments, and elastic transforms using the Albumentations library to enhance model generalization.



Data Splitting:

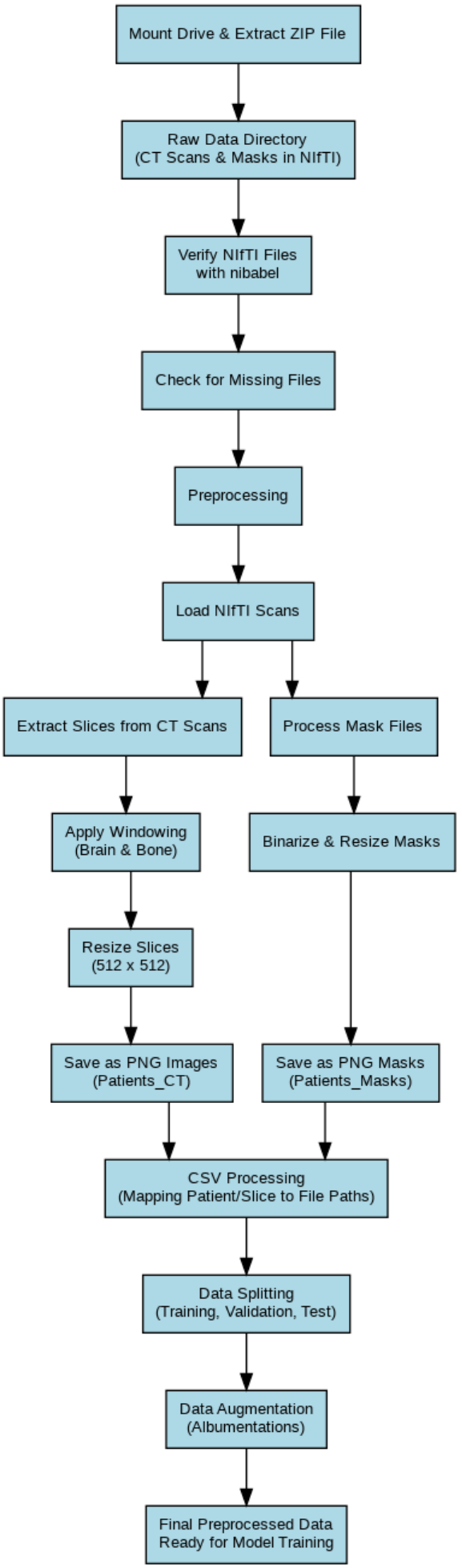
The dataset was split into training (80%), validation (20%), and a separate test set. Care was taken to ensure patient-wise independence.

Training set shape: (2235, 11)

Validation set shape: (559, 11)

Test set set shape: (20, 11)

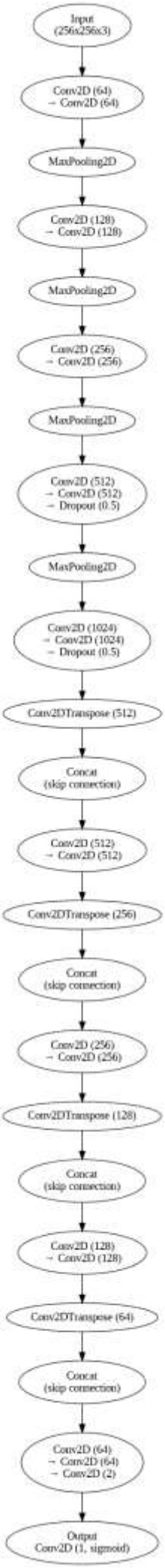
This figure shows the overall process pipeline of the Data and Processing step >>>



Methodology

Model Architecture:

A U-Net convolutional neural network was designed for 256×256 RGB images. The encoder extracts features using its successive convolution and max-pooling layers, while the decoder employs transposed convolutions and concatenation with high-resolution features to produce pixel-level segmentation. Proper action of how CNN works. This figure on the right side shows the U-Net architecture with key layers




```
# Encoder (Downsampling)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(inputs)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(pool1)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(pool2)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

conv4 = Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(pool3)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv4)
drop4 = Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
```

```
# Decoder (Upsampling)
up6 = Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(drop5)
merge6 = concatenate([drop4, up6], axis=3)
conv6 = Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(merge6)
conv6 = Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv6)

up7 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(conv6)
merge7 = concatenate([conv3, up7], axis=3)
conv7 = Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(merge7)
conv7 = Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv7)

up8 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv7)
merge8 = concatenate([conv2, up8], axis=3)
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(merge8)
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv8)

up9 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv8)
merge9 = concatenate([conv1, up9], axis=3)
conv9 = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(merge9)
conv9 = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv9)
conv9 = Conv2D(2, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv9)

conv10 = Conv2D(1, (1, 1), activation='sigmoid')(conv9)
```

Two dropout layers at the bottleneck were added to reduce overfitting.

```
# Bottleneck
conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(pool4)
conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same', kernel_initializer=he_normal())(conv5)
drop5 = Dropout(0.5)(conv5)
```

The final layer uses a sigmoid activation function for binary segmentation (hemorrhage vs. non-hemorrhage).

Additionally, an initialized of Binary cross-entropy was used as the loss function, and the Adam optimizer was chosen with an initial learning rate of 1e-4.

Training Metrics:

Accuracy and loss curves were monitored over epochs. Early stopping, model checkpointing, and learning rate reduction (ReduceLROnPlateau) were implemented to prevent overfitting.

Augmentation Integration:

On-the-fly augmentation was applied to both images and masks during training, which improved model robustness. Training was conducted on a GPU-enabled platform using TensorFlow and Keras.

For details refer table:

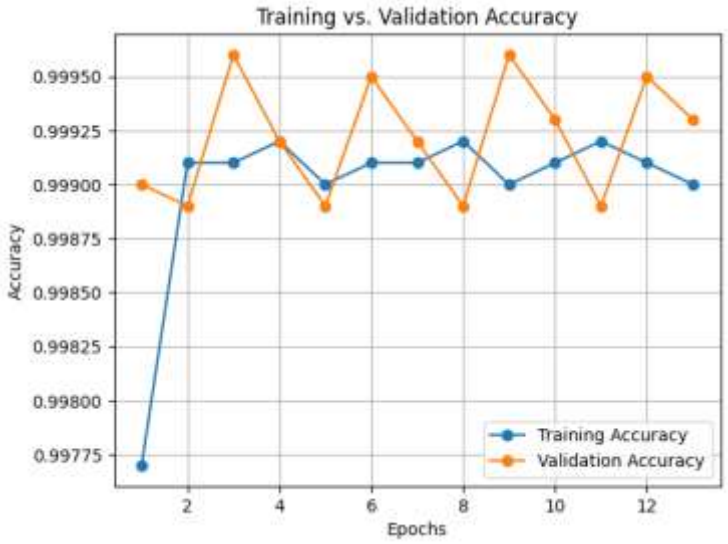
	Parameter	Value	Description
0	Batch Size	10	Number of samples per training batch
1	Learning Rate	1e-4	Initial learning rate for the Adam optimizer
2	Encoder Dropout Rate	0.5	Applied after the conv4 block in the encoder
3	Bottleneck Dropout Rate	0.5	Applied after the conv5 block in the bottleneck
4	Epochs	50	Maximum number of training epochs
5	Early Stopping Patience	10	Number of epochs with no improvement before stopping
6	LR Reduction Factor	0.5	Factor by which the learning rate is reduced
7	LR Reduction Patience	5	Number of epochs with no improvement before reduction

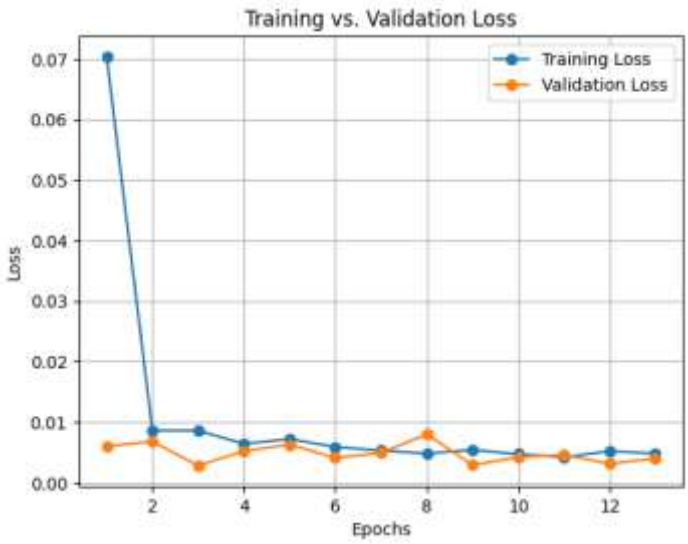
Results

```
Epoch 1/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 668ms/step - accuracy: 0.9977 - loss: 0.0705WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
280/280 ━━━━━━━━━━━━━━━━━━━━ 265s 740ms/step - accuracy: 0.9977 - loss: 0.0704 - val_accuracy: 0.9990 - val_loss: 0.0060 - learning_rate: 1.0000e-04
Epoch 2/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 153s 547ms/step - accuracy: 0.9991 - loss: 0.0086 - val_accuracy: 0.9989 - val_loss: 0.0068 - learning_rate: 1.0000e-04
Epoch 3/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 528ms/step - accuracy: 0.9991 - loss: 0.0086WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
280/280 ━━━━━━━━━━━━━━━━━━━━ 173s 620ms/step - accuracy: 0.9991 - loss: 0.0086 - val_accuracy: 0.9996 - val_loss: 0.0028 - learning_rate: 1.0000e-04
Epoch 4/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 151s 540ms/step - accuracy: 0.9992 - loss: 0.0064 - val_accuracy: 0.9992 - val_loss: 0.0052 - learning_rate: 1.0000e-04
Epoch 5/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 152s 542ms/step - accuracy: 0.9990 - loss: 0.0072 - val_accuracy: 0.9989 - val_loss: 0.0063 - learning_rate: 1.0000e-04
Epoch 6/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 152s 543ms/step - accuracy: 0.9991 - loss: 0.0059 - val_accuracy: 0.9995 - val_loss: 0.0041 - learning_rate: 1.0000e-04
Epoch 7/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 202s 722ms/step - accuracy: 0.9991 - loss: 0.0053 - val_accuracy: 0.9992 - val_loss: 0.0049 - learning_rate: 1.0000e-04
Epoch 8/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 531ms/step - accuracy: 0.9992 - loss: 0.0048
Epoch 8: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
280/280 ━━━━━━━━━━━━━━━━━━━━ 202s 722ms/step - accuracy: 0.9992 - loss: 0.0048 - val_accuracy: 0.9989 - val_loss: 0.0080 - learning_rate: 1.0000e-04
Epoch 9/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 152s 544ms/step - accuracy: 0.9990 - loss: 0.0054 - val_accuracy: 0.9996 - val_loss: 0.0029 - learning_rate: 5.0000e-05
Epoch 10/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 152s 544ms/step - accuracy: 0.9991 - loss: 0.0047 - val_accuracy: 0.9993 - val_loss: 0.0042 - learning_rate: 5.0000e-05
Epoch 11/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 152s 544ms/step - accuracy: 0.9992 - loss: 0.0042 - val_accuracy: 0.9989 - val_loss: 0.0046 - learning_rate: 5.0000e-05
Epoch 12/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 202s 722ms/step - accuracy: 0.9991 - loss: 0.0052 - val_accuracy: 0.9995 - val_loss: 0.0031 - learning_rate: 5.0000e-05
Epoch 13/50
280/280 ━━━━━━━━━━━━━━━━━━━━ 0s 534ms/step - accuracy: 0.9990 - loss: 0.0048
Epoch 13: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
280/280 ━━━━━━━━━━━━━━━━━━━━ 202s 722ms/step - accuracy: 0.9990 - loss: 0.0048 - val_accuracy: 0.9993 - val_loss: 0.0040 - learning_rate: 5.0000e-05
```

Early Stopping on 13th epoch

The model showed steady convergence, with training accuracy exceeding 99% and validation accuracy hovering around 99% after 12 epochs. The loss curves for both training and validation indicate minimal overfitting.





After training, the best model achieved a validation loss of ~0.003 and accuracy of nearly 99%, indicating excellent performance on the held-out validation set.

Test Evaluation and Statistical Analysis

Test Set Performance:

On the test set (20 samples), the model produced segmented masks that were thresholded (using a threshold of 0.7) and compared against ground truth masks.

Quantitative Metrics: Key metrics such as segmentation area (pixel count of the predicted hemorrhage region) were computed for each test sample.

Statistical Testing: Two-sample t-tests were performed comparing segmented areas between

Group1 (Non-hemorrhage cases): 10 samples

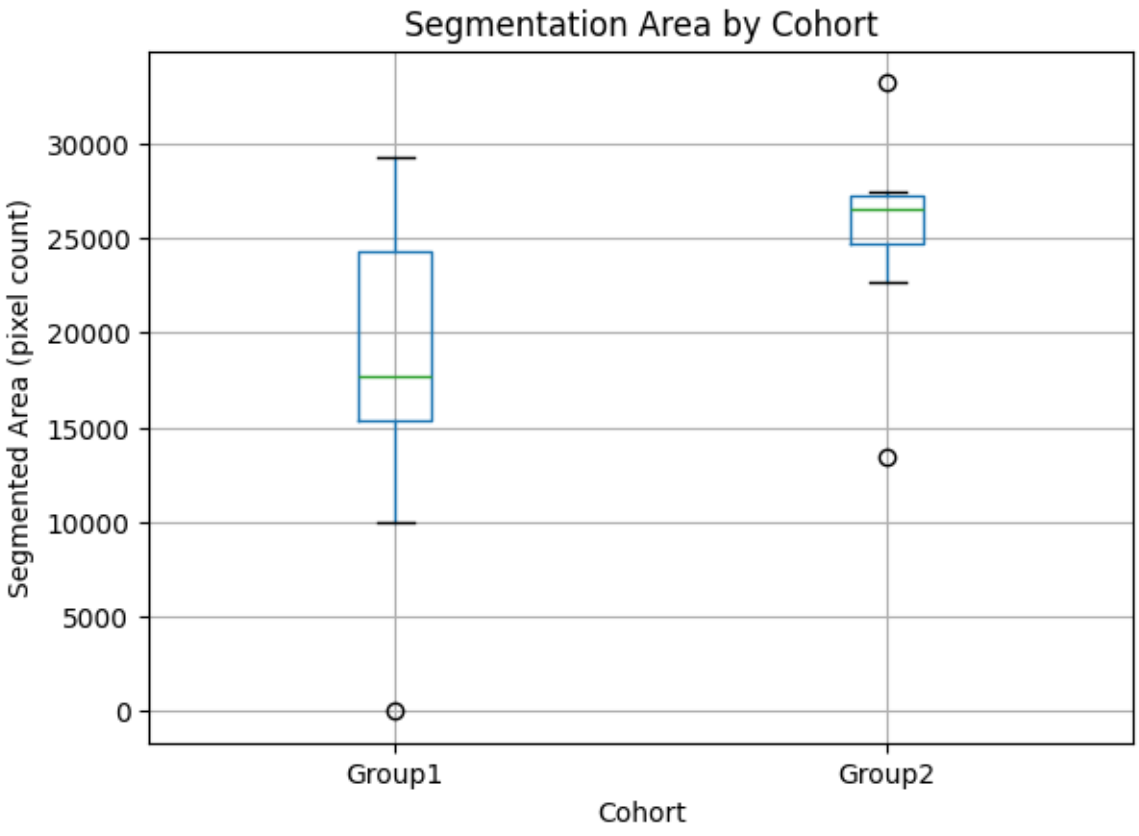
Group2 (Hemorrhage cases): 10 samples

The t-test yielded a t-statistic of -2.266 and a p-value of 0.036, indicating a statistically significant difference between groups.

Additionally, a one-way ANOVA (after simulating a third group) returned an F-statistic of 3.503 with a p-value of 0.053, further supporting the significance of the differences observed.

Test	Statistic	p-value	Group Sizes
Two-sample t-test	-2.26607	0.03601	Group1: 10, Group2: 10
One-way ANOVA	3.50296	0.05322	Group1: 10, Group2: 10

Boxplots of segmented area by cohort (Group1 vs. Group2) illustrate the distribution differences.



The high training and validation accuracy shows us that the U-Net model was effectively abled to learn to segment brain regions and identify hemorrhagic abnormalities. The statistical tests t-test and ANOVA, done on them confirm that the segmentation area differs significantly between hemorrhage and non-hemorrhage cases, supporting the relevance of the model's outputs.

Limitations of this model is that our dataset is of small size size and diversity with a limited number of patient scans and potential scanner variability, further validation on a larger, multi-center dataset will be better is my take on this. One more thing to note is that since slices were processed independently, incorporating 3D context might further enhance segmentation accuracy. Covering all these in a short period is not that much feasible but maybe we can work on the enhancement of this model using the provided details above.

Conclusion

This project was on developing an open-source AI system that automatically segments brain images and detects hemorrhages using a U-Net model. The model performed very well—it achieved high accuracy and was able to clearly differentiate between hemorrhage and non-hemorrhage cases with statistically significant results. When we looked at the segmentation overlays, the quality was impressive, which shows that this framework could really help in clinical settings. In the future, we'll work on gathering more diverse data and incorporating 3D analysis to make the detection even better.

Note:

Generating heatmaps, volumetric plots made the system crash and I tried by importing the model to a new notebook and plotting but never worked. Still working on it, I included Visualization in segmentation area instead.

Colab link