Automated Tumor Detection using DenseNet201

- Vaishnavi Chilumuru
- Bharath
- Surya Kiran Varma Vegesna
- Srujani Mareddy
- Naveen Kumar Reddy Singam

Georgia State University | J. MACK ROBINSON COLLEGE OF BUSINESS

Robinson

Problem Statement

1. **Time-Consuming Manual Analysis**: Brain tumors are typically diagnosed through **MRI** scans, which require detailed analysis by skilled neuroradiologists. This process is labor-intensive and time-consuming, potentially delaying diagnosis and treatment.

2. **Complexity of MRI Data**: MRI produces highly detailed images showing diverse tissue contrasts, making the manual segmentation of brain tumors a complex task that demands high precision and expert knowledge.

3. **Need for Automation**: Automating the process of tumor detection and segmentation can significantly enhance the efficiency and accuracy of brain tumor diagnosis, providing timely support for treatment planning and potentially improving patient outcomes.

Methodology

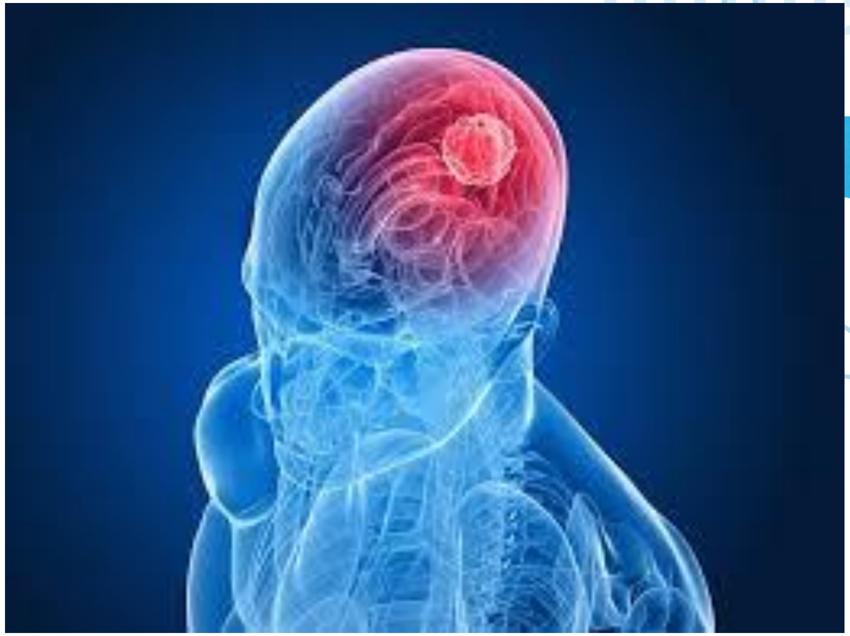
Automated Tumor Detection and Localization

- **Objective**: Develop a deep learning system to automatically detect and localize brain tumors in MRI scans.

- **Key Tasks**: Train a classifier to identify tumor presence and a segmentation model to outline tumor regions.

- **Model Training**: Utilize DenseNet201 for tumor presence classification and ResUNet for tumor localization.

- **Medical Utility**: Provide accurate tumor detection and localization aids in early diagnosis and treatment planning.

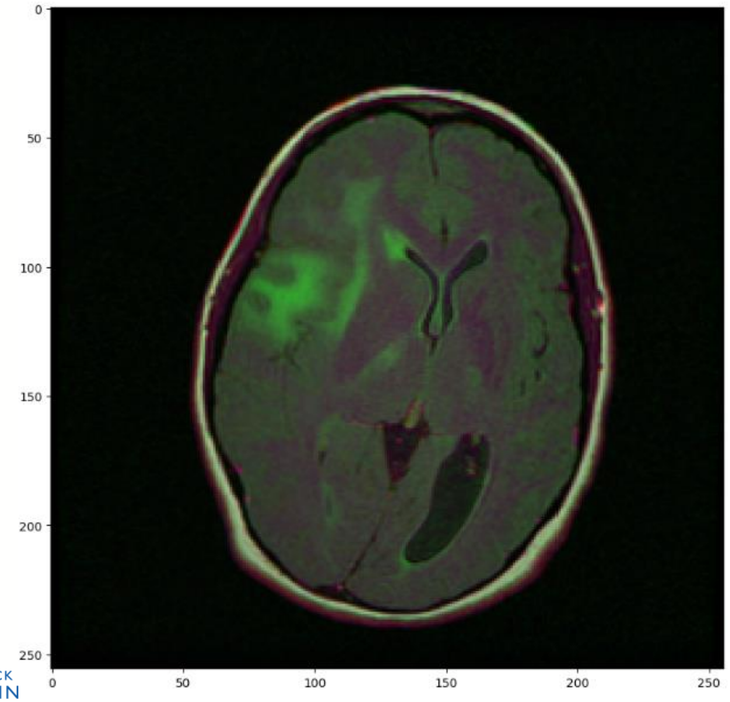


Data Set

Two important variables in the DataFrame:

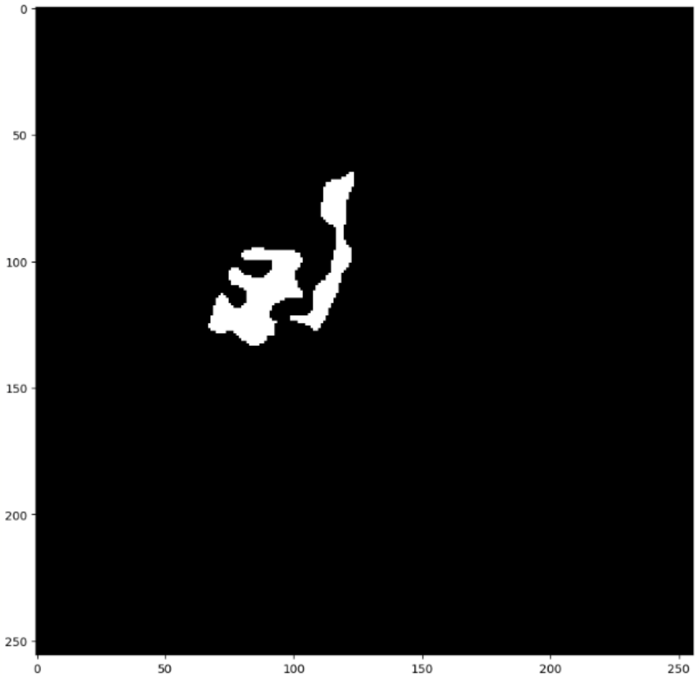
- image_path

'TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif'



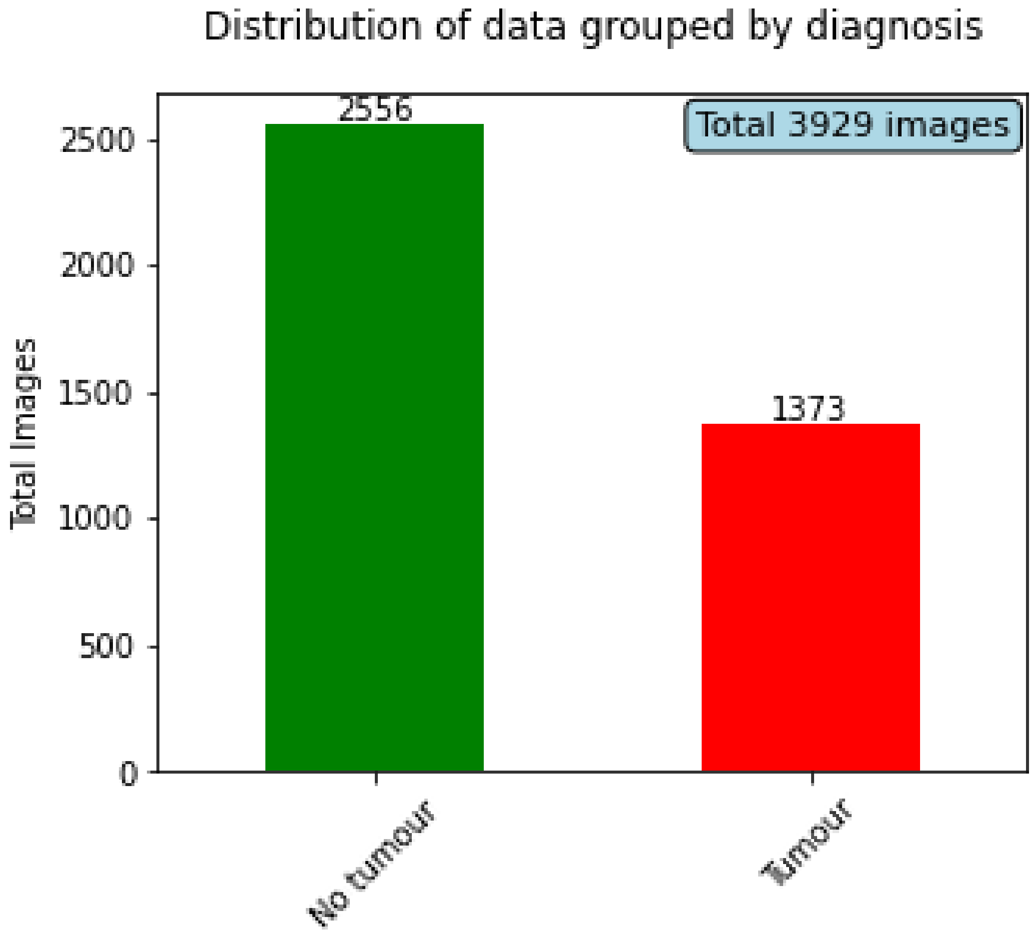
- mask_path

'TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1_mask.tif'



Data Visualization

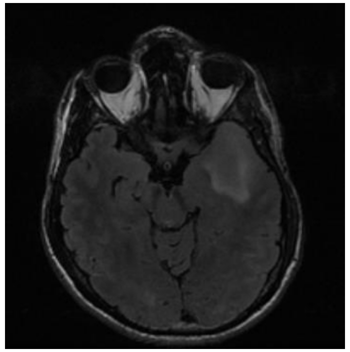
- Investigated and quantified the number of images in the dataset with and without associated masks, providing insights into tumor prevalence.
- Illustrated the dataset's tumor presence distribution, revealing that approximately 34.95% of images contains tumors, while around 65.05% do not.



Data Preparation

Image Data Pre-processing

Dataset

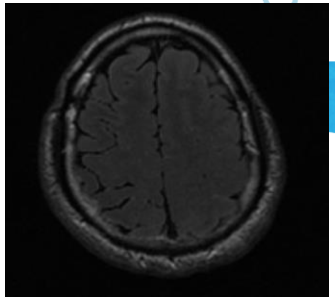


Original Batch of images

ImageDataGenerator Function

```
# create a image generator
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Create a data generator which scales the data from 0 to 1 and makes validation split of 0.1
datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.1)
```

Prepares the data for the model by rescaling the pixel values and setting up training, validation, and test generators



Preprocessed train dataset of images

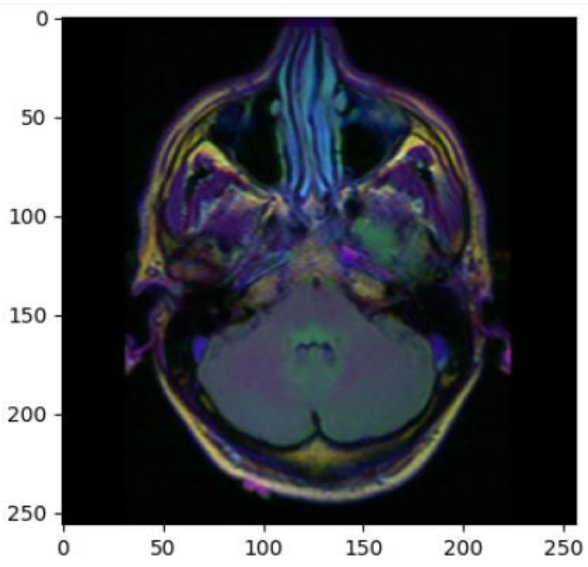
Visualization of MRI Scans with Segmentation Masks

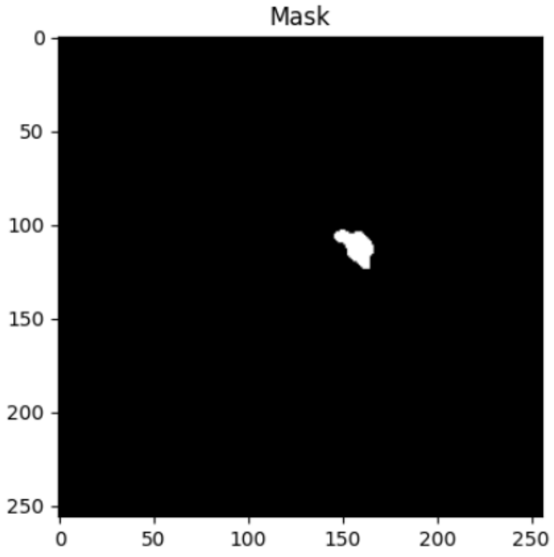
```
count = 0
fig, axs = plt.subplots(10, 3, figsize = (20, 40))
for i in range(len(brain_df)):
  if brain_df['mask'][i] ==1 and count <10:
    img = io.imread(brain_df.image_path[i])
    axs[count][0].title.set_text('Brain MRI')
    axs[count][0].imshow(img)

    mask = io.imread(brain_df.mask_path[i])
    axs[count][1].title.set_text('Mask')
    axs[count][1].imshow(mask, cmap = 'gray')


    img[mask == 255] = (255, 0, 0)
    axs[count][2].title.set_text('MRI with Mask')
    axs[count][2].imshow(img)
    count+=1

fig.tight_layout()
```

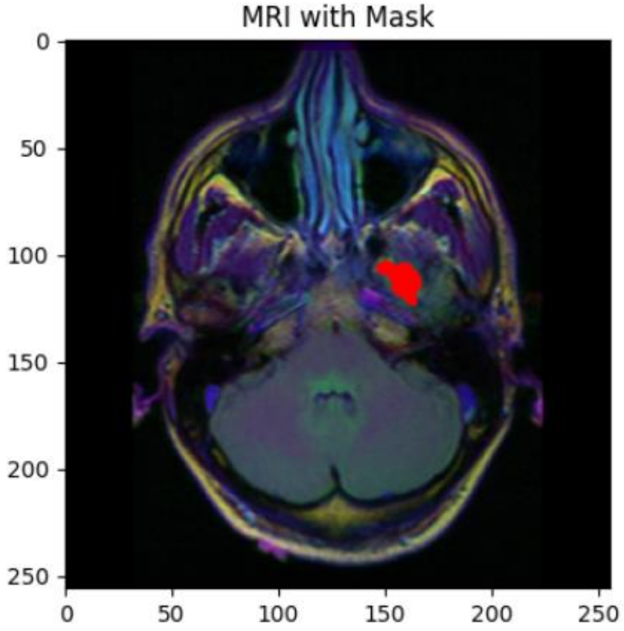
- This script visualizes MRI brain scans alongside their corresponding segmentation masks to highlight areas identified as tumors."

- It processes MRI scans with masks, displaying the original scan, the segmentation mask, and the scan with the mask overlaid in red."

Process and results of Segmentation MRI Scans



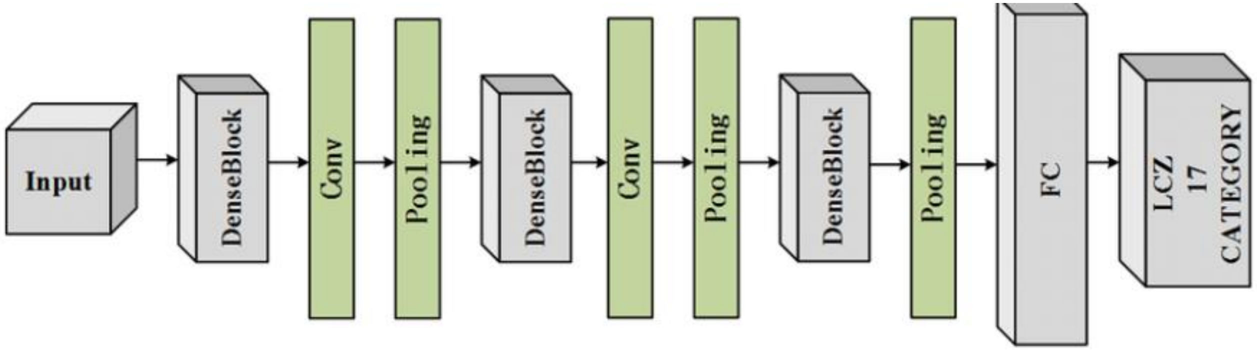


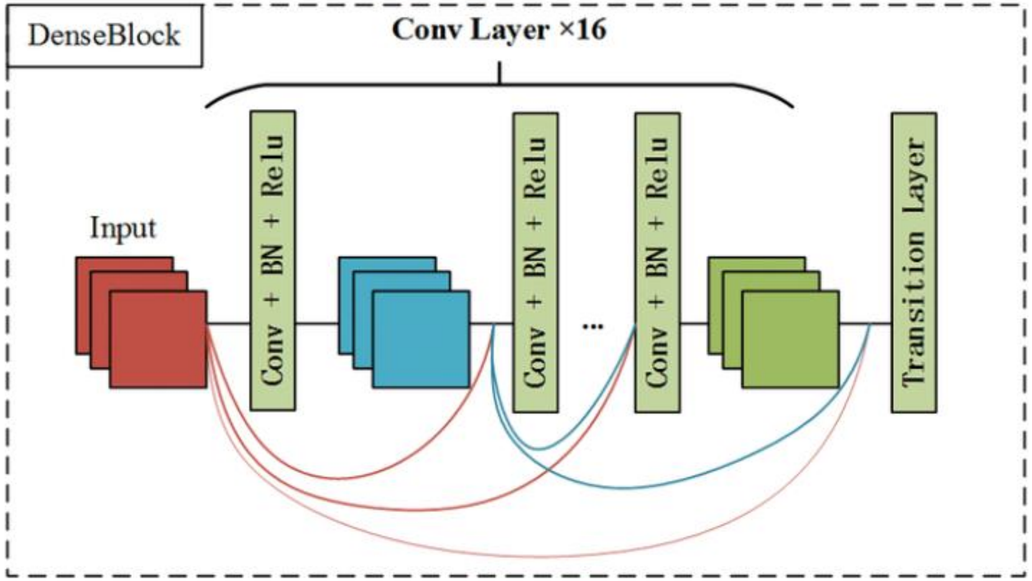


Data Pre-processing

1. Dropping Pateint_ID column
2. Convert mask column to string format
3. data into train and test data
4. create a image generator
5. Create a data generator for test images
6. DenseNet201 base model
7. freeze the model weights
8. classification head to the base model
9. Model Complilation

DenseNet Architecture





Model Evaluation Metrics

High Precision & Recall: Model demonstrates excellent accuracy with 97-98% precision, meaning most positive predictions are correct, and 95-99% recall, indicating it successfully identifies the majority of actual cases.

Robust F1-Score: Achieves an F1-score of 0.96-0.98, reflecting a strong balance between precision and recall, essential for medical diagnostic reliability.

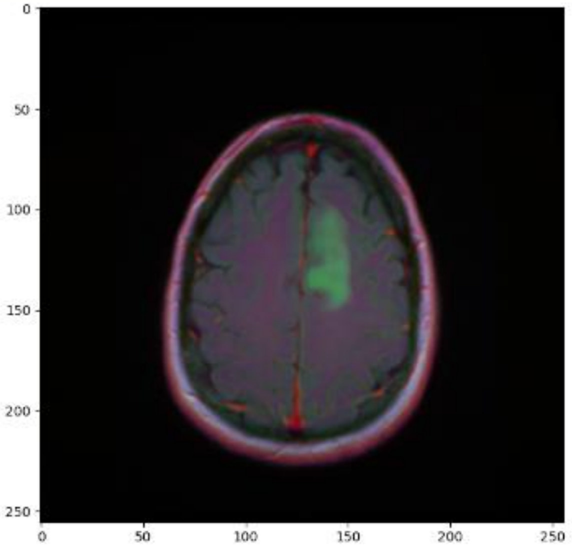
Consistent Performance: Weighted averages for precision, recall, and F1-score are all at 0.97, showing consistent model performance across classes, crucial in imbalanced datasets.

```
[ ]  from sklearn.metrics import classification_report

     report = classification_report(original, predict, labels = [0,1])
     print(report)
```

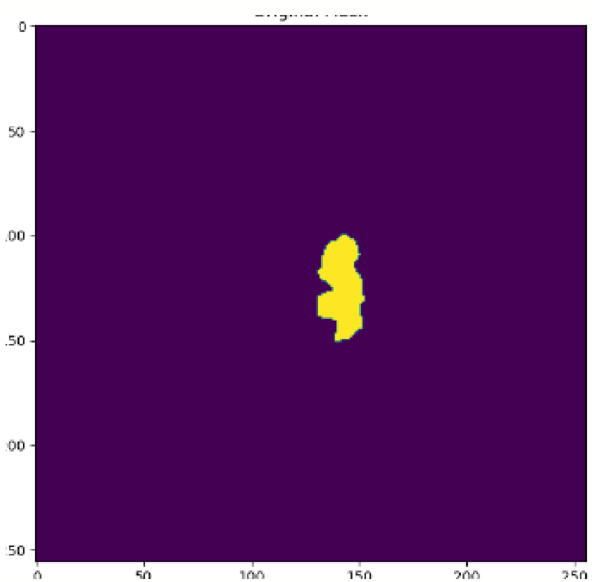
	precision	recall	f1-score	support
0	0.97	0.99	0.98	497
1	0.98	0.95	0.96	287
micro avg	0.97	0.97	0.97	784
macro avg	0.98	0.97	0.97	784
weighted avg	0.97	0.97	0.97	784

Results

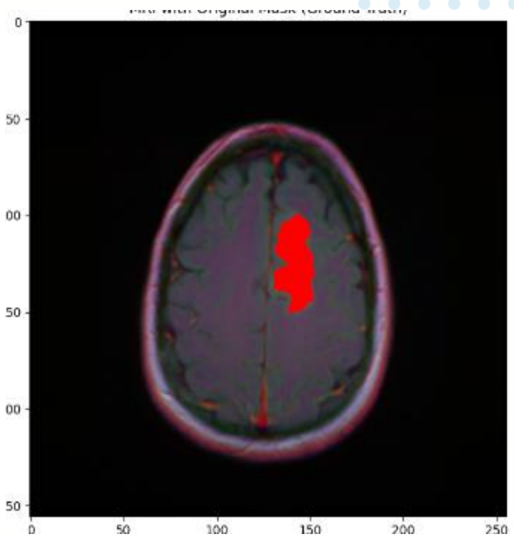
Brain Mask



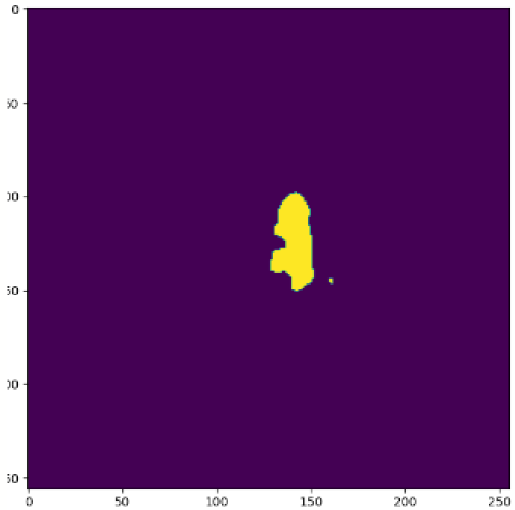
Original Mask



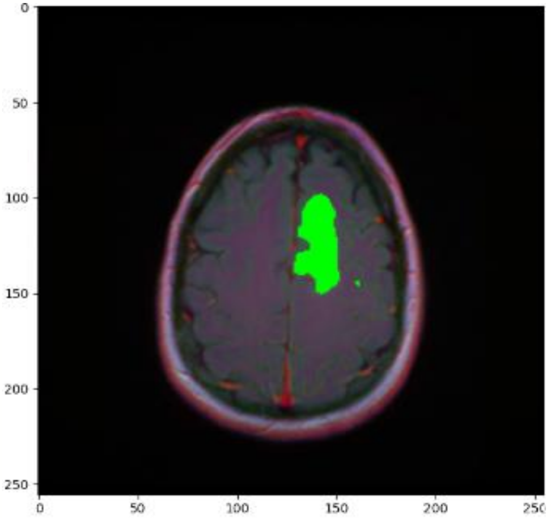
MRI with original mask (Ground Truth)



AI Predicted Mask



MRI with AI Predictive Mask



Questions

Flappy Bird

Game Description :

- **Type**: Simplified version of 'Flappy Bird' video game.
- **Objective**: Navigate a bird avatar through pipe gaps without collisions as the game speeds up**.**

Hypothesis for AI Agent :

- **Action Rule**: Flap wings based on bird position, pipe gap proximity, and game speed.
- **Reward Mechanism**: Binary reward for successful gap passage; negative reward for collisions or altitude errors.

Reinforcement learning model architecture

Input Layer: Receives game state image (bird, pipes, background) as pixel matrix for feature detection.

Convolutional Layers: These layers use filters to detect features such as the positions of pipes relative to the bird. They help identify important visual elements crucial for decision-making.

Flatten Layer: Converts the output of convolutional layers into a one-dimensional array, preparing it for processing by dense layers.

Dense Layers: Analyze the flattened features to determine the best actions based on the game state. These layers use ReLU activation to handle non-linear data effectively.

Output Layer: Predicts Q-values for actions like "flap" or "do nothing." Depending on the number of actions, it uses either sigmoid (two actions) or softmax (more than two actions) activation to output probabilities.