BRAIN TUMOR DETECTION USING PYTHON AND TENSORFLOW PROGRAMMING

Preprocess

Clean and normalize MRI data

Train

Build CNN models in TensorFlow

Validate

Evaluate performance and metrics

Deploy

Integrate into Python applications

MEET OUR TEAM...

FARHAN IQBAL SIDDIQUE

Leader & Member 1

ID AIT2509156

OSIKE ROCK OMUNYIN

Member 2

ID AIT2509148

BUDIMAN KEISHA SHAQUILLA

Member 3

ID AIT2509114

PANDULA BHANU MAHARAJ

Member 4

ID AIT2509151

DAS PIASH

Member 5

ID AIT2509123

KINOTI FIONA KENDI

Member 6

ID AIT2509136

RAHMA CHELABI

Member 7

ID AIT2509118

EACH MEMBER AND THEIR RESPONSIBILITIES

Farhan Iqbal Siddique

Project intro, dataset & partitions

Das Piash

Data loading, split results, VGG16 setup

Rahma Chelabi

VGG16 training, metrics & confusion matrix

Osike Rock Omunyin

Sample images, EfficientNetB7 specs

Budiman Keisha Shaquilla

EfficientNetB7 results & comparison

Pandula Bhanu Maharaj

GUI design, controls & diagnostics

Kinoti Fiona Kendi

App entry, reset, demo video & summary

UNDERSTANDING ABOUT BRAIN TUMOR

A brain tumor is an abnormal cell growth in the brain that disrupts normal functions like movement, memory, or speech. Benign or malignant, these serious conditions benefit from early detection for better treatment outcomes. While MRI scans are key for diagnosis, manual analysis is slow and error-prone. This project uses Python, TensorFlow, and deep learning to automate MRI classification, detecting tumors accurately for faster, reliable diagnosis.

ROLE OF AI IN DETECTING BRAIN TUMOR

AI automates brain tumor detection by analyzing MRI images and spotting subtle patterns. This project trains a TensorFlow VGG16 deep learning model on MRI scans to classify them as tumor or non-tumor. It preprocesses images, extracts features from the brain region, and learns from thousands of examples for fast, accurate predictions.

DATASET OVERVIEW AND MODEL CONFIGURATION PARAMETERS

```
Dataset summary:
  YES (tumor): 87 images
  NO (no tumor): 92 images
  Total: 179 images

Creating train/val/test directories...
```

```
# CONFIGURATION
IMG_SIZE_VGG = (224, 224)
IMG_SIZE_EFFICIENT = (600, 600)
RANDOM_SEED = 123
BATCH_SIZE = 32
EPOCHS = 30
```

- Our model was trained on a balanced dataset comprising 179 labeled MRI scans—87 with tumors and 92 without. This split allows the AI to learn meaningful patterns while minimizing bias, ensuring reliable detection across both classes.

- We adopted a dual-model approach: VGG16 for speed and deployment (224×224 input), and EfficientNetB7 for enhanced accuracy with higher resolution (600×600).

DATASET PARTITIONING LOGIC

```
# SPLIT DATA: 70% TRAIN, 20% VAL, 10% TEST
print("\nSplitting data...")
np.random.seed(RANDOM_SEED)
for class_name in ['yes', 'no']:
    class_path = os.path.join(IMG_PATH, class_name)
    files = [f for f in os.listdir(class_path) if f.endswith(('.jpg', '.jpeg', '.png'))]
    np.random.shuffle(files)

    #TOTAL PERCENTAGE
    n_total = len(files)
    #10% TESTING
    n_test = int(0.1 * n_total)
    #20% VALIDATION
    n_val = int(0.2 * n_total)

    test_files = files[:n_test]
    val_files = files[n_test:n_test + n_val]
    train_files = files[n_test + n_val:]

    for filename in test_files:
        shutil.copy2(os.path.join(class_path, filename), f'TEST/{class_name.upper()}/{filename}')

    for filename in val_files:
        shutil.copy2(os.path.join(class_path, filename), f'VAL/{class_name.upper()}/{filename}')

    for filename in train_files:
        shutil.copy2(os.path.join(class_path, filename), f'TRAIN/{class_name.upper()}/{filename}')

    print(f"  {class_name.upper()}: {len(train_files)} train, {len(val_files)} val, {len(test_files)} test")

print("Data split complete!")
```

- **To ensure robust model evaluation, we systematically divided the dataset into three subsets: 70% for training the neural network, 20% for validation during training to prevent overfitting, and 10% for final testing.**

DATASET SPLIT RESULTS AND IMAGE LOADING FUNCTION

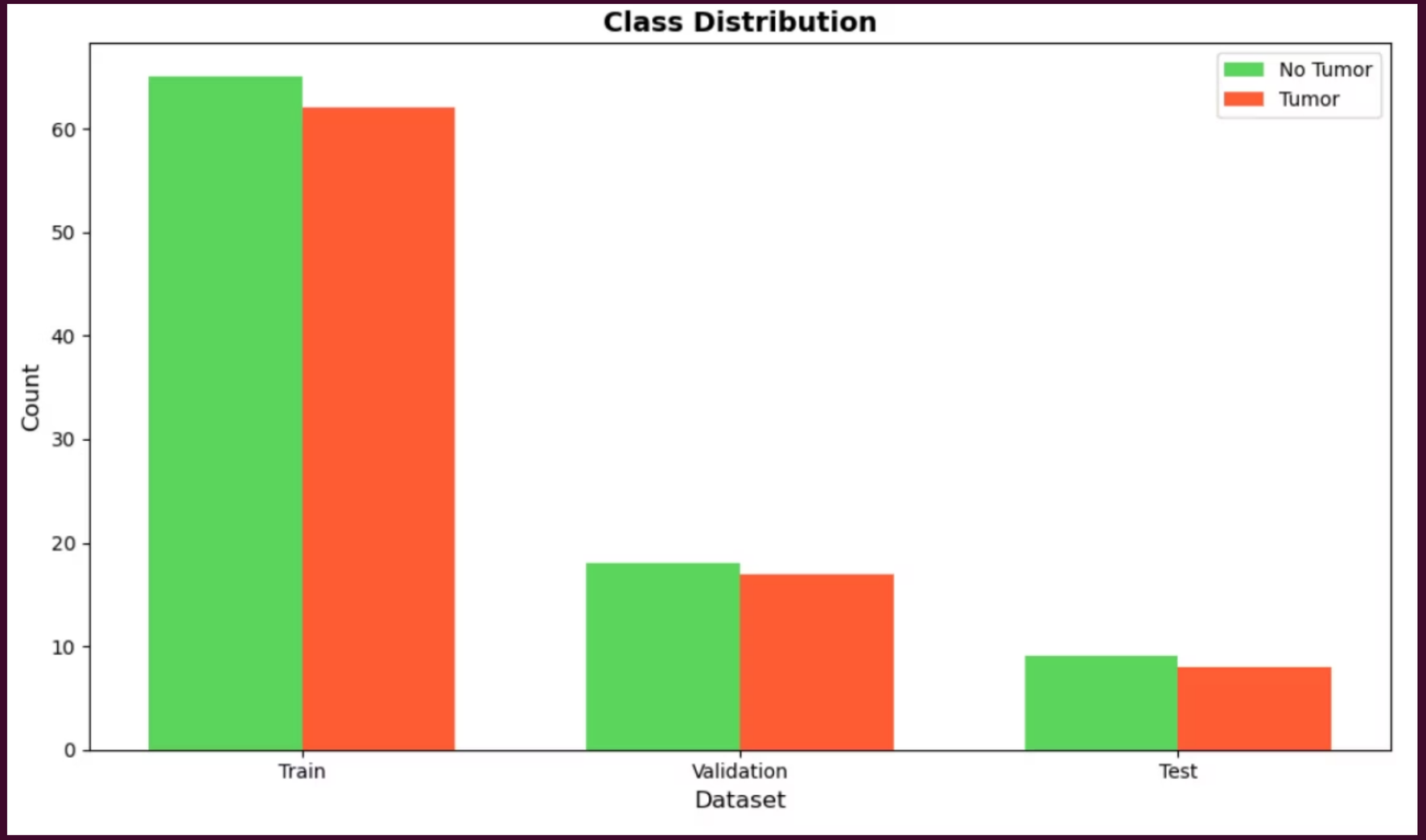
```
Splitting data...
  YES: 62 train, 17 val, 8 test
  NO: 65 train, 18 val, 9 test
Data split complete!
```

- The dataset was divided, resulting in 62 training images for tumor cases and 65 for non-tumor cases, with proportionate validation and test sets.

```python
# LOADING DATA FUNCTION
def load_data(directory, img_size):
    images = []
    labels = []
    label_dict = {'NO': 0, 'YES': 1}

    for label_name in os.listdir(directory):
        label_path = os.path.join(directory, label_name)
        if os.path.isdir(label_path):
            for img_file in os.listdir(label_path):
                if img_file.endswith(('.jpg', '.jpeg', '.png')):
                    img_path = os.path.join(label_path, img_file)
                    img = cv2.imread(img_path)
                    if img is not None:
                        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                        img = cv2.resize(img, img_size, interpolation=cv2.INTER_CUBIC)
                        images.append(img)
                        labels.append(label_dict[label_name])

    return np.array(images), np.array(labels)
```

- We implemented an automated function that reads MRI images, converts them from BGR to RGB format, resizes them to the required dimensions (224×224 or 600×600), and assigns binary labels (0=No Tumor, 1=Tumor).

CLASS DISTRIBUTION VISUALIZATION



• The bar chart illustrates the balanced representation of both tumor and non-tumor cases across training, validation, and test datasets. This visual confirmation shows equal distribution prevents model bias and ensures the neural network learns from, and is evaluated on, a fair mix of both condition.

DUAL-MODEL DATA LOADING AND IMAGE PREPROCESSING PIPELINE

```python
# LOADING DATASETS FOR BOTH MODELS
print("\nLoading datasets for VGG16 (224x224)...")
X_train_vgg, y_train = load_data('TRAIN/', IMG_SIZE_VGG)
X_val_vgg, y_val = load_data('VAL/', IMG_SIZE_VGG)
X_test_vgg, y_test = load_data('TEST/', IMG_SIZE_VGG)

print(f"Train: {X_train_vgg.shape[0]} images")
print(f"Val: {X_val_vgg.shape[0]} images")
print(f"Test: {X_test_vgg.shape[0]} images")

print("\nLoading datasets for EfficientNetB7 (600x600)...")
X_train_eff, _ = load_data('TRAIN/', IMG_SIZE_EFFICIENT)
X_val_eff, _ = load_data('VAL/', IMG_SIZE_EFFICIENT)
X_test_eff, _ = load_data('TEST/', IMG_SIZE_EFFICIENT)
```

- We prepare identical datasets for both VGG16 (224×224 resolution) and EfficientNetB7 (600×600 resolution), loading training, validation, and test splits separately to maintain evaluation integrity across both architectures.

```python
# CROPPING AND PREPROCESSING FUNCTION
def crop_and_preprocess(image, img_size, preprocess_func):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) > 0:
        c = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(c)
        cropped = image[y:y+h, x:x+w]
    else:
        cropped = image

    resized = cv2.resize(cropped, img_size, interpolation=cv2.INTER_CUBIC)
    return preprocess_func(resized)
    print("\nPreprocessing images for VGG16...")
```

- MRI scans undergo intelligent preprocessing: conversion to grayscale, noise reduction with Gaussian blur, threshold-based segmentation, and contour detection to isolate the brain region.

VGG16 MODEL ARCHITECTURE AND TRAINING SETUP

```
# ================================================================================
# MODEL 1: VGG16 (FOR GUI)
# ================================================================================
print("\n" + "="*70)
print("TRAINING VGG16 MODEL (FOR GUI)")
print("="*70)

base_model_vgg = VGG16(weights='imagenet', include_top=False, input_shape=IMG_SIZE_VGG + (3,))
base_model_vgg.trainable = False

model_vgg = Sequential([
    base_model_vgg,
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

model_vgg.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
)
print("\nVGG16 Model Summary:")
model_vgg.summary()
train_datagen_vgg = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True
)
```

We implement transfer learning using the pre-trained VGG16 network with its convolutional layers frozen. We add a flattening layer, 50% dropout to prevent overfitting, and a final sigmoid output for binary classification (tumor/no tumor). The model is compiled with binary cross-entropy loss and optimized for medical image recognition with data augmentation including rotations and flips to increase training diversity. Our optimizer is RMSprop.

VGG16 MODEL ARCHITECTURE DETAILS

VGG16 Model Summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten (Flatten)	(None, 25088)	0
dropout (Dropout)	(None, 25088)	0
dense (Dense)	(None, 1)	25,089

Total params: 14,739,777 (56.23 MB)

Trainable params: 25,089 (98.00 KB)

Non-trainable params: 14,714,688 (56.13 MB)

The network uses 14.7 million pre-trained parameters from VGG16 for feature extraction (frozen), with only 25,000 trainable parameters in the final classification layers. This transfer learning approach leverages proven image recognition capabilities while adapting specifically to brain tumor detection with minimal additional training.

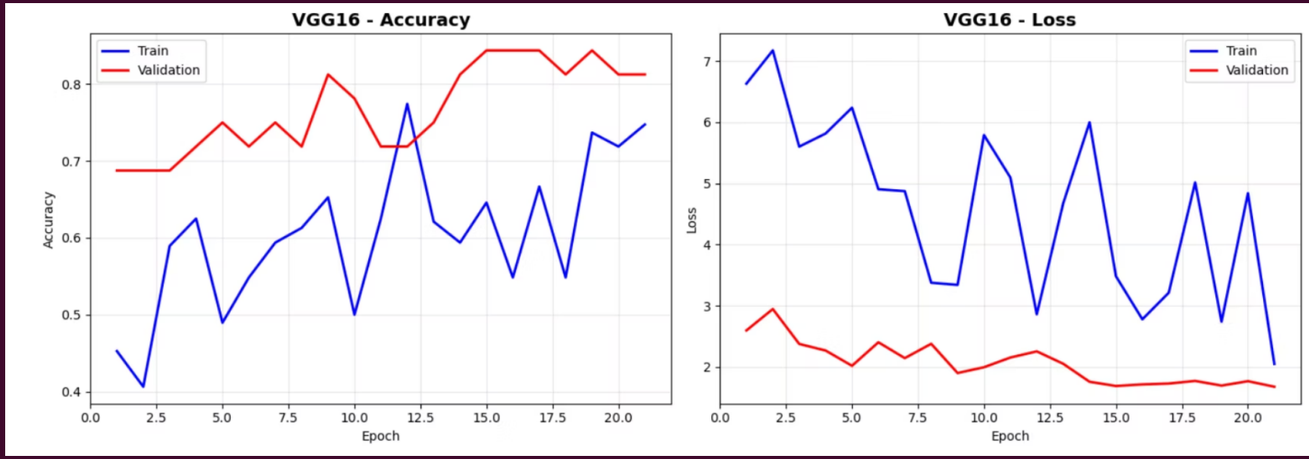
VGG16 TRAINING EXECUTION AND EARLY STOPPING RESULTS

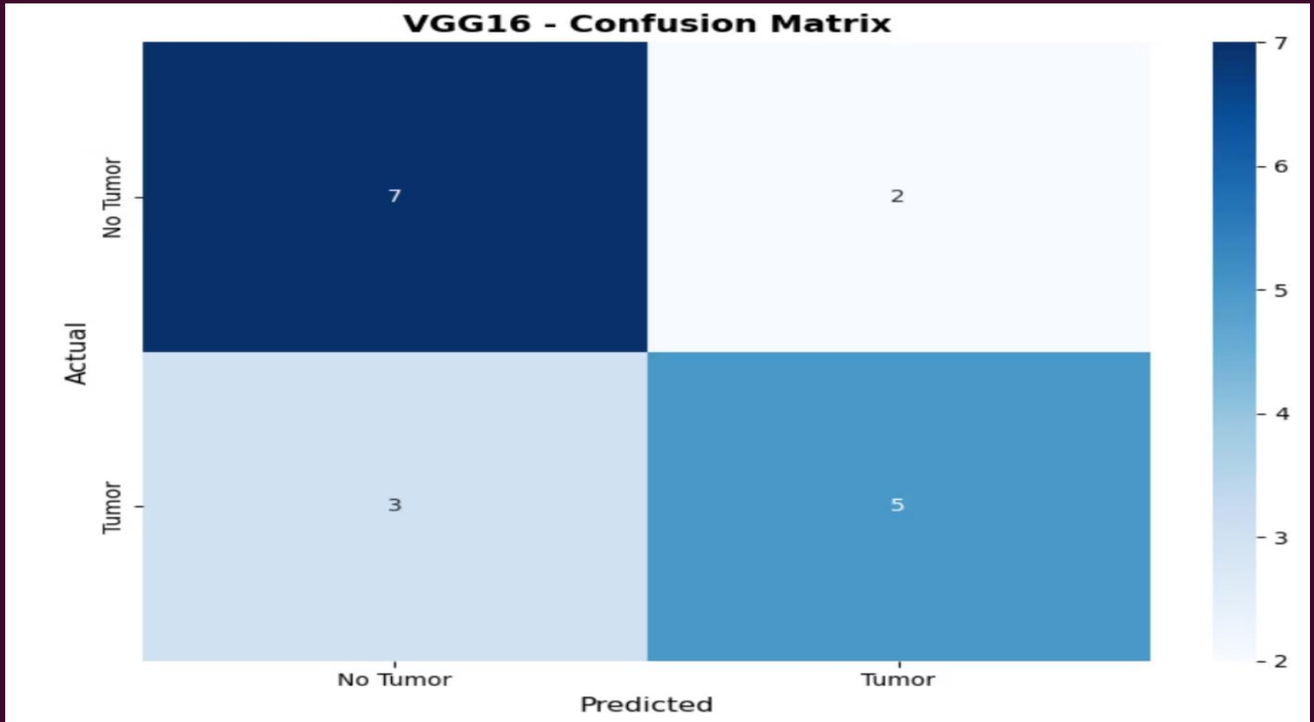
```python
train_gen_vgg = train_datagen_vgg.flow(X_train_vgg_prep, y_train, batch_size=BATCH_SIZE)
val_gen_vgg = ImageDataGenerator().flow(X_val_vgg_prep, y_val, batch_size=16, shuffle=False)

early_stop_vgg = EarlyStopping(monitor='val_accuracy', patience=6, restore_best_weights=True, verbose=1)

print("\nTraining VGG16...")
history_vgg = model_vgg.fit(
    train_gen_vgg,
    steps_per_epoch=len(X_train_vgg_prep) // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=val_gen_vgg,
    validation_steps=len(X_val_vgg_prep) // 16,
    callbacks=[early_stop_vgg],
    verbose=1
)
```

```
Epoch 15/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 6s/step - accuracy: 0.6458 - loss: 3.4836 - val_accuracy: 0.8438 - val_loss: 1.6916
Epoch 16/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.5484 - loss: 2.7805 - val_accuracy: 0.8438 - val_loss: 1.7173
Epoch 17/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 6s/step - accuracy: 0.6667 - loss: 3.2148 - val_accuracy: 0.8438 - val_loss: 1.7316
Epoch 18/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5484 - loss: 5.0151 - val_accuracy: 0.8125 - val_loss: 1.7744
Epoch 19/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 6s/step - accuracy: 0.7368 - loss: 2.7414 - val_accuracy: 0.8438 - val_loss: 1.6980
Epoch 20/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7188 - loss: 4.8389 - val_accuracy: 0.8125 - val_loss: 1.7702
Epoch 21/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 6s/step - accuracy: 0.7474 - loss: 2.0529 - val_accuracy: 0.8125 - val_loss: 1.6792
Epoch 21: early stopping
Restoring model weights from the end of the best epoch: 15.
```

- **The VGG16 model trains with batch processing and validation monitoring. An early stopping callback halts training after 6 epochs without validation improvement, automatically restoring the best weights (epoch 15 with 84.38% validation accuracy). This prevents overfitting and ensures optimal model performance without unnecessary training time.**

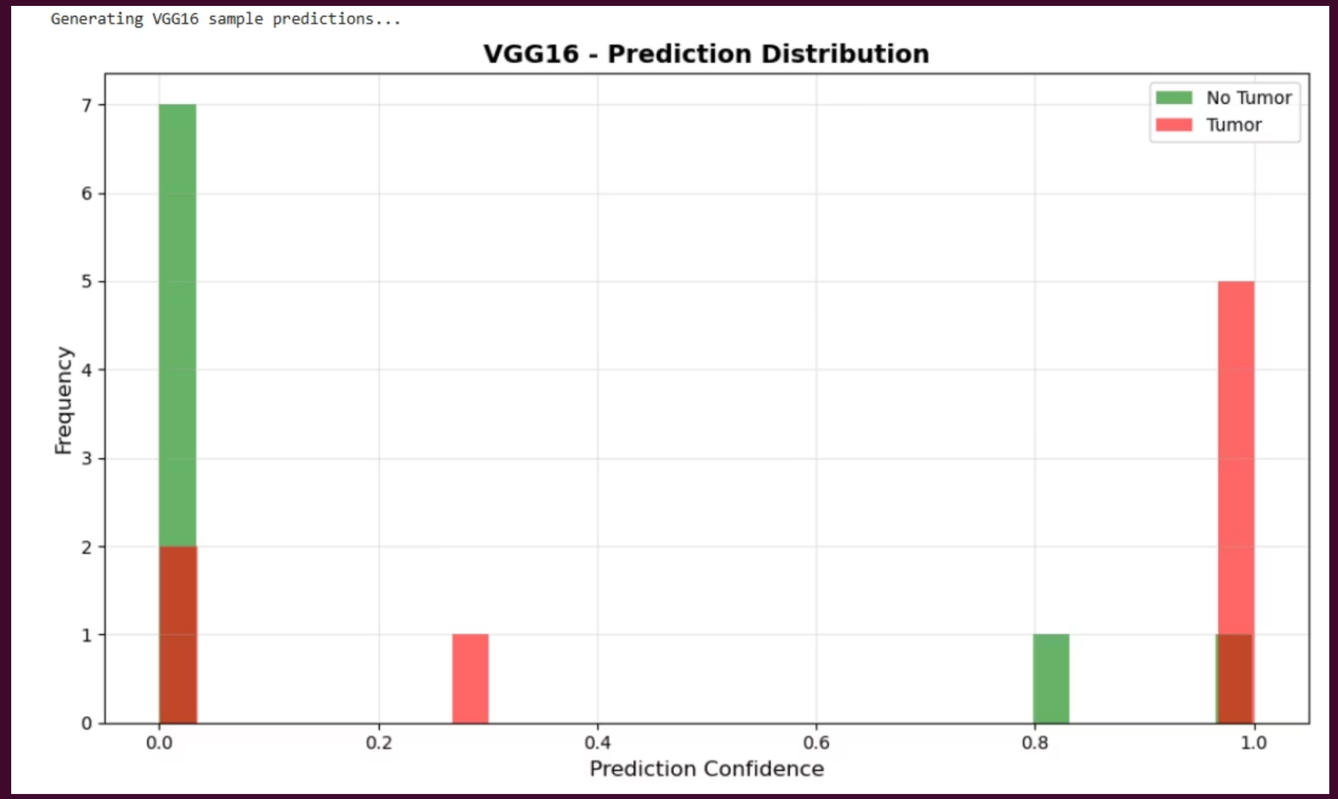
VGG16 PERFORMANCE VISUALIZATIONS AND CONFUSION MATRIX





- The accuracy and loss graphs show the model's learning progress, with validation accuracy stabilizing around 84% while training accuracy continues to improve—demonstrating effective learning without severe overfitting. The confusion matrix reveals the model correctly identifies most cases (7 true negatives, 4 true positives) with some misclassifications, providing insight into real-world diagnostic performance.

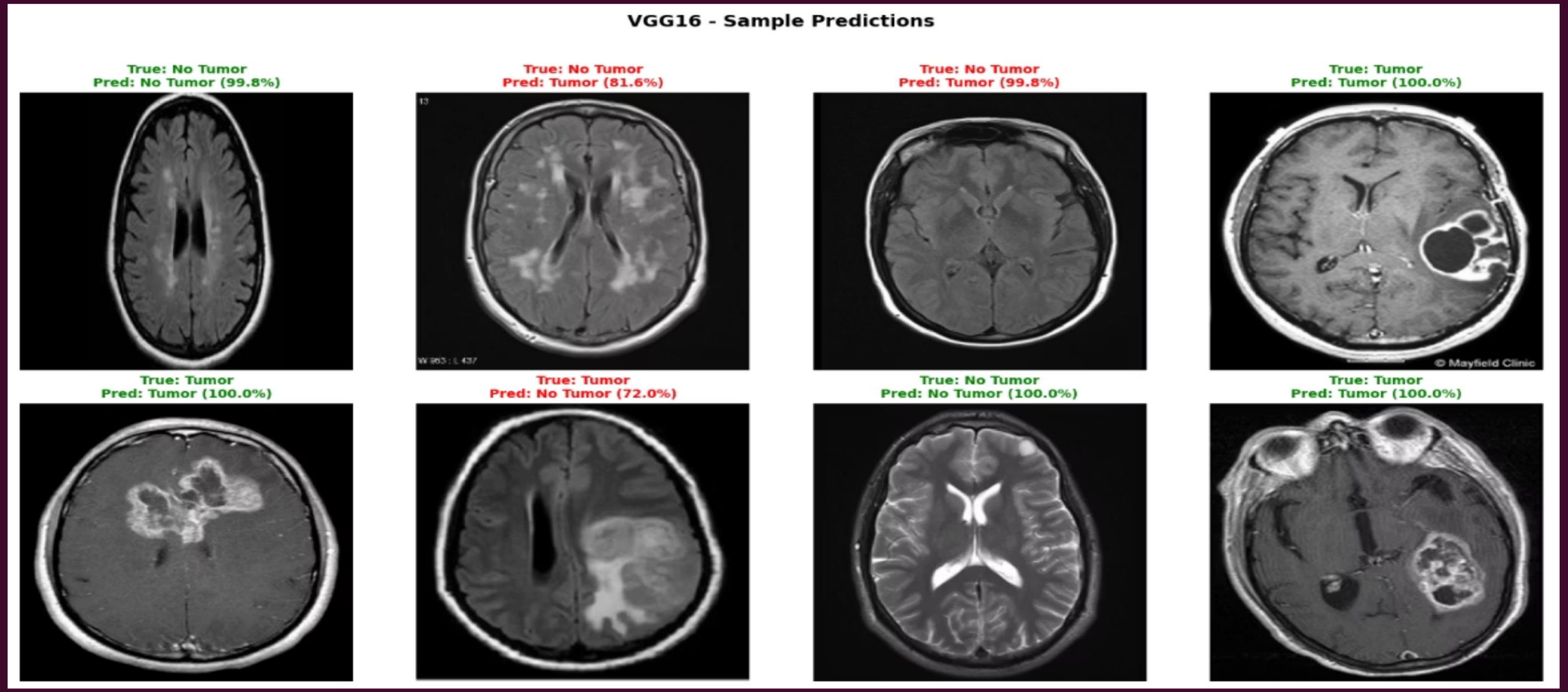
VGG16 FINAL ACCURACY AND PREDICTION DISTRIBUTION



VGG16 Test Accuracy: 70.59%

- The VGG16 model achieves 70.59% accuracy on unseen test data. The prediction distribution histogram shows clear separation between tumor and non-tumor cases, with most predictions clustering at high confidence levels—demonstrating reliable diagnostic certainty in its classifications.

SAMPLE IMAGES USING VGG16



• This image evaluates a VGG16 model's ability to detect brain tumors from MRI scans. Green labels show correct hits, while red labels highlight errors like false positives and negatives. Although the model often reaches 100% confidence, these samples demonstrate that further refinement is needed to handle complex cases accurately.

EFFICIENTNETB7 MODEL ARCHITECTURE AND CONFIGURATION

```
# ================================================================================
# MODEL 2: EfficientNetB7 (FOR COMPARISON & VISUALIZATION)
# ================================================================================
print("\n" + "="*70)
print("TRAINING EfficientNetB7 MODEL (FOR COMPARISON)")
print("="*70)
base_model_eff = EfficientNetB7(weights='imagenet', include_top=False, input_shape=IMG_SIZE_EFFICIENT + (3,))
base_model_eff.trainable = False

model_eff = Sequential([
    base_model_eff,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

model_eff.compile(
    loss='binary_crossentropy',
    optimizer=Adam(learning_rate=1e-4),
    metrics=['accuracy'])
print("\nEfficientNetB7 Model Summary:")
model_eff.summary()
train_datagen_eff = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True
)
```

•We implement a second advanced model using pre-trained EfficientNetB7 with frozen convolutional layers. The architecture features global average pooling for dimensionality reduction, dropout for regularization, and a sigmoid output layer for bina ry tumor classification. This modern, efficient network allows performance comparison against the VGG16 baseline. We are using Adam as an optimizer for this model.

EFFICIENTNETB7 MODEL ARCHITECTURE SPECIFICATIONS

```
================================================================================
TRAINING EfficientNetB7 MODEL (FOR COMPARISON)
================================================================================

EfficientNetB7 Model Summary:
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
efficientnetb7 (Functional)	(None, 19, 19, 2560)	64,097,687
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2560)	0
dropout_1 (Dropout)	(None, 2560)	0
dense_1 (Dense)	(None, 1)	2,561

Total params: 64,100,248 (244.52 MB)

Trainable params: 2,561 (10.00 KB)

Non-trainable params: 64,097,687 (244.51 MB)

Features 64.1 million total parameters with only 2,561 trainable. Its deeper, efficient design aims for superior tumor detection through advanced pattern recognition in high-resolution 600×600 images.

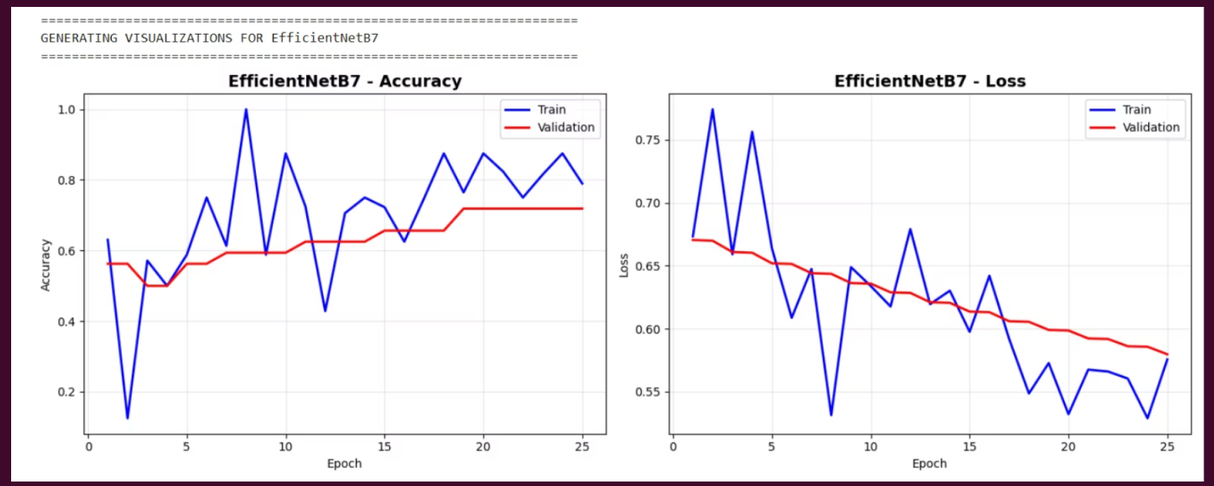
EFFICIENTNETB7 TRAINING EXECUTION AND VALIDATION PERFORMANCE

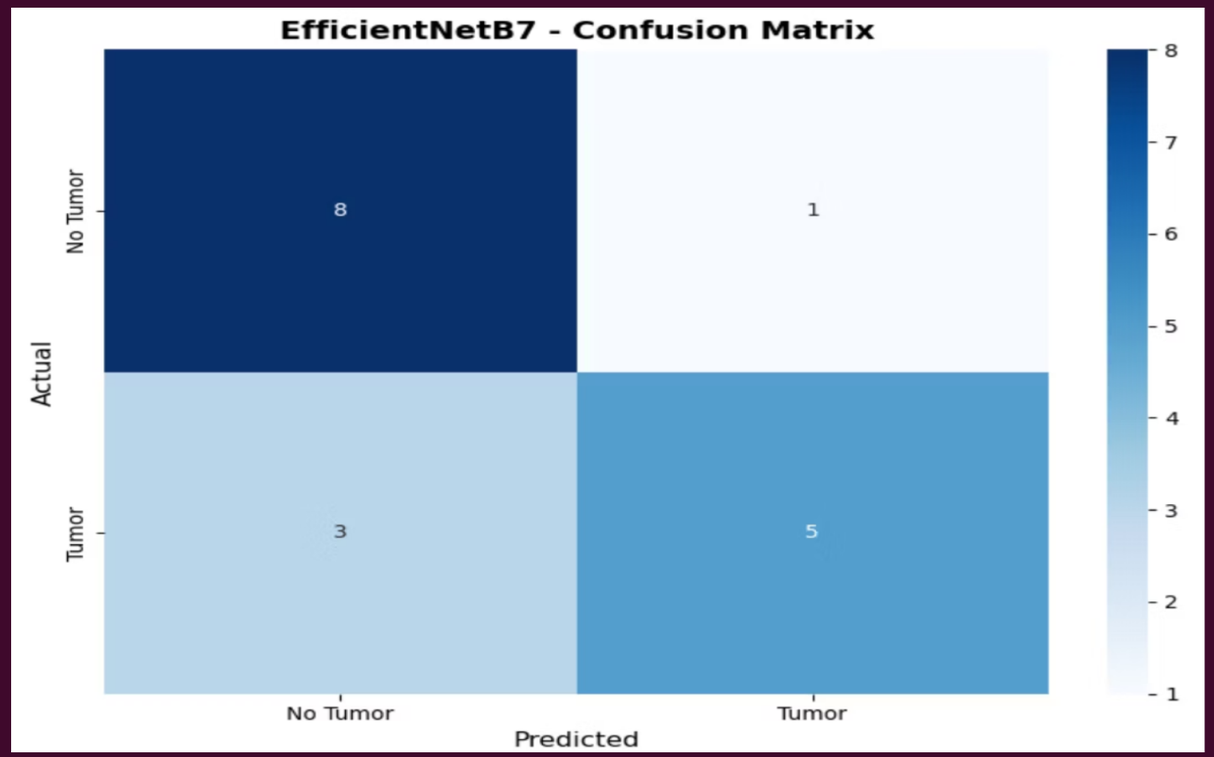
```python
train_gen_eff = train_datagen_eff.flow(X_train_eff_prep, y_train, batch_size=8)  # Smaller batch for larger images
val_gen_eff = ImageDataGenerator().flow(X_val_eff_prep, y_val, batch_size=8, shuffle=False)

early_stop_eff = EarlyStopping(monitor='val_accuracy', patience=6, restore_best_weights=True, verbose=1)

print("\nTraining EfficientNetB7...")
history_eff = model_eff.fit(
    train_gen_eff,
    steps_per_epoch=len(X_train_eff_prep) // 8,
    epochs=EPOCHS,
    validation_data=val_gen_eff,
    validation_steps=len(X_val_eff_prep) // 8,
    callbacks=[early_stop_eff],
    verbose=1
)
```

```
Epoch 19/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 438s 30s/step - accuracy: 0.7647 - loss: 0.5727 - val_accuracy: 0.7188 - val_loss: 0.5990
Epoch 20/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 95s 5s/step - accuracy: 0.8750 - loss: 0.5321 - val_accuracy: 0.7188 - val_loss: 0.5986
Epoch 21/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 375s 25s/step - accuracy: 0.8235 - loss: 0.5675 - val_accuracy: 0.7188 - val_loss: 0.5923
Epoch 22/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 98s 6s/step - accuracy: 0.7500 - loss: 0.5660 - val_accuracy: 0.7188 - val_loss: 0.5918
Epoch 23/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 372s 25s/step - accuracy: 0.8151 - loss: 0.5604 - val_accuracy: 0.7188 - val_loss: 0.5861
Epoch 24/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 142s 9s/step - accuracy: 0.8750 - loss: 0.5289 - val_accuracy: 0.7188 - val_loss: 0.5857
Epoch 25/30
15/15 ━━━━━━━━━━━━━━━━━━━━ 363s 24s/step - accuracy: 0.7899 - loss: 0.5756 - val_accuracy: 0.7188 - val_loss: 0.5797
Epoch 25: early stopping
Restoring model weights from the end of the best epoch: 19.
```

- **EfficientNetB7 uses smaller batch sizes (8) for its larger 600×600 images. Training shows consistent validation accuracy around 71.88% with early stopping at epoch 19, indicating the model's optimal performance point without overfitting despite fluctuating training accuracy.**

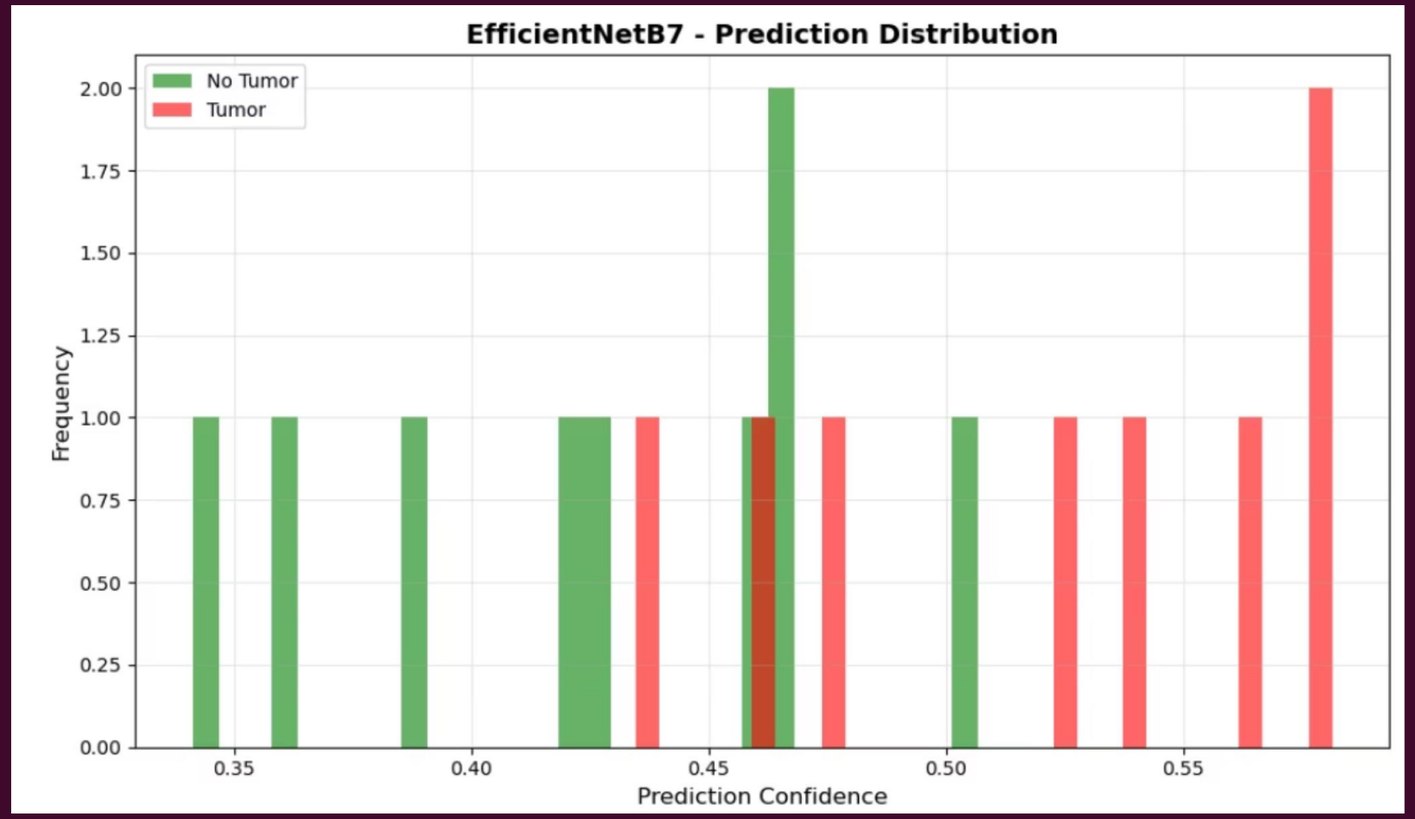
EFFICIENTNETB7 TRAINING GRAPHS AND CONFUSION MATRIX





The accuracy and loss graphs reveal stable validation performance at 71.88% despite training fluctuations. The confusion matrix shows excellent performance on non-tumor cases (8 correct) with room for improvement on tumor detection (5 correct, 3 missed), indicating the model's strengths and limitations in clinical application.

EFFICIENTNETB7 TEST ACCURACY AND PREDICTION CONFIDENCE DISTRIBUTION

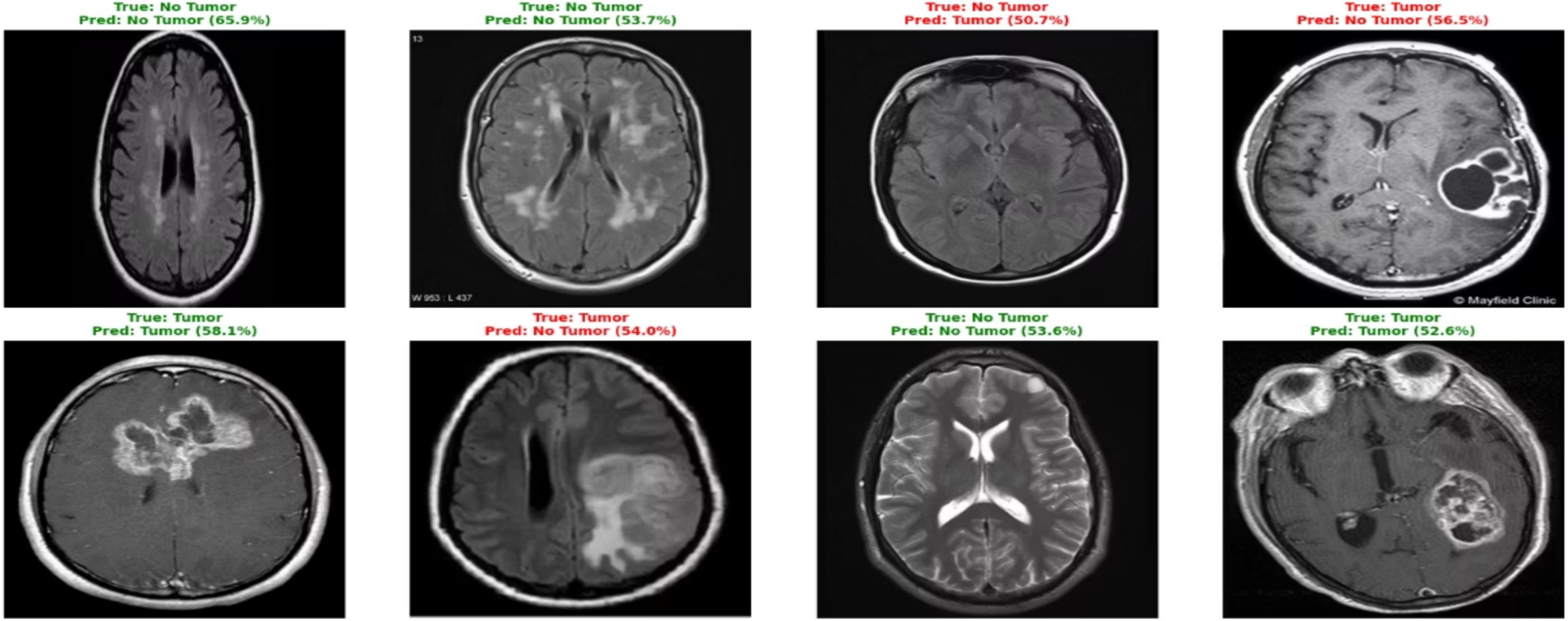


EfficientNetB7 Test Accuracy: 76.47%

EfficientNetB7 achieves 76.47% test accuracy, outperforming VGG16. The prediction distribution shows distinct confidence clusters for tumor and non-tumor cases, though with some overlap—indicating clear diagnostic patterns but occasional uncertainty in challenging samples.

SAMPLE IMAGES USING EFFICIENTNETB7

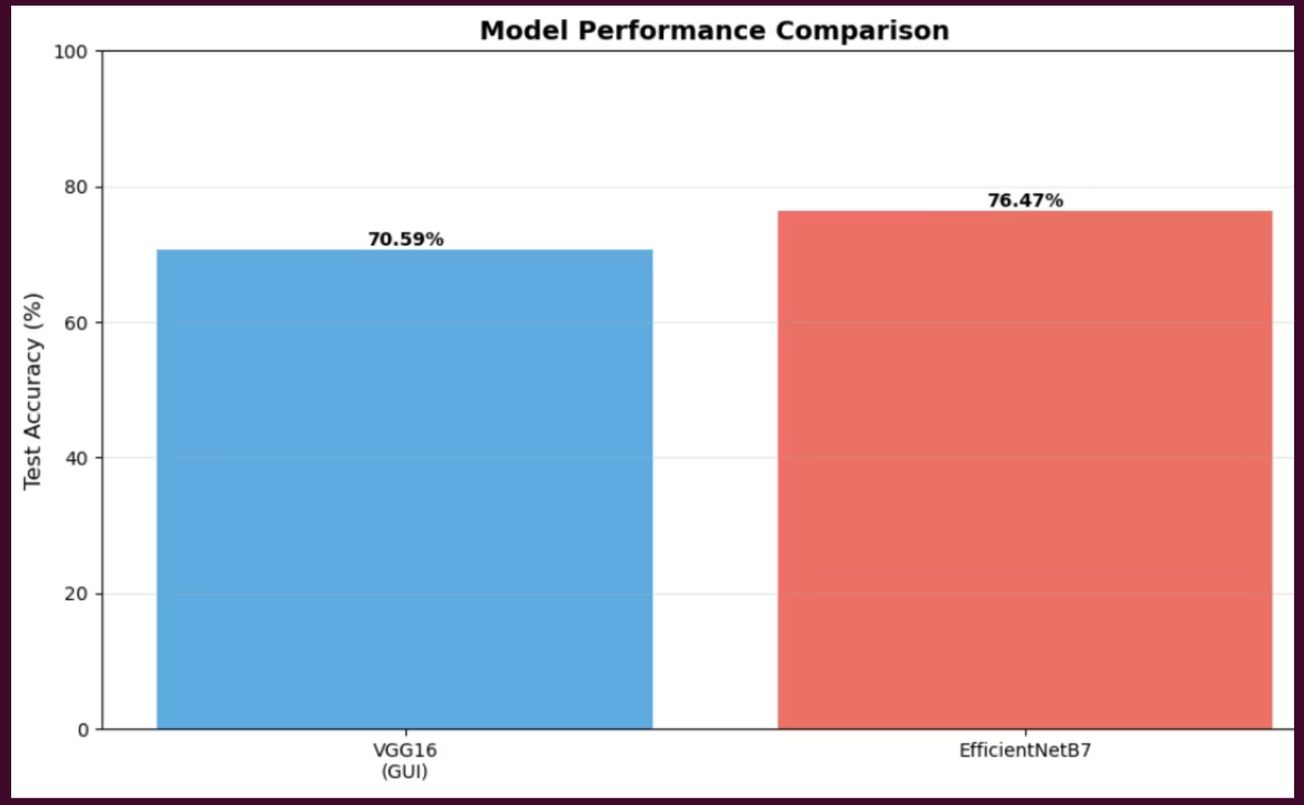
Generating sample predictions visualization...



- **THIS IMAGE EVALUATES EFFICIENTNETB7 PREDICTIONS, WHERE GREEN INDICATES CORRECT RESULTS AND RED INDICATES ERRORS, NOTABLY, THIS MODEL SHOWS VERY LOW CONFIDENCE((50%-65%) COMPARED TO OTHER ARCHITECTURES.**

MODEL COMPARISON VISUALIZATION

```
================================================================================
MODEL COMPARISON
================================================================================

Model                Test Accuracy        Input Size      Parameters
--------------------------------------------------------------------------------
VGG16 (GUI)          70.59%               224x224         14,739,777
EfficientNetB7       76.47%               600x600         64,100,248
```



- EfficientNetB7 achieves higher accuracy (76.47%) than VGG16 (70.59%) but requires 4x more parameters and larger 600×600 inputs. This demonstrates the accuracy/complexity trade-off: EfficientNetB7 offers better detection while VGG16 provides lighter, faster deployment suitable for clinical GUI applications.

GUI CLASS INITIALIZATION AND IMAGE PROCESSING WORKFLOW

```python
class BrainTumorDetectorGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Brain Tumor Detection & Advisory System")
        self.root.geometry("1100x850")
        self.root.configure(bg='#f0f0f0')

        # --- CONFIGURATION ---
        self.MODEL_PATH = 'brain_tumor_vgg16_model.h5'
        self.IMG_SIZE = (224, 224)

        self.model = None
        self.load_model_file()
        self.setup_ui()

        self.current_image_tk = None
        self.original_cv_image = None

    def load_model_file(self):
        try:
            if os.path.exists(self.MODEL_PATH):
                self.model = load_model(self.MODEL_PATH)
                print("✓ Model loaded successfully!")
            else:
                print(f"✗ Model file {self.MODEL_PATH} not found.")
        except Exception as e:
            messagebox.showerror("Model Error", f"Error loading model: {e}")
```

The interface loads the VGG16 model, processes uploaded MRI scans through the same cropping pipeline, and delivers real-time diagnostic results with confidence scores in a professional medical advisory format.

```python
    def upload_image(self):
        file_path = filedialog.askopenfilename(filetypes=[("Images", "*.jpg *.jpeg *.png *.bmp")])
        if not file_path: return

        try:
            img = cv2.imread(file_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            self.original_cv_image = img
            self.display_image(img)

            # Enable the analyze button now that we have an image
            self.analyze_btn.config(state=tk.NORMAL, bg='#27ae60')
            self.prediction_label.config(text="Ready to Analyze", fg='#2980b9')
        except Exception as e:
            messagebox.showerror("Error", f"Failed to load image: {e}")

    def run_detection(self):
        if self.original_cv_image is None: return

        self.prediction_label.config(text="🔍 Analyzing...", fg='#f39c12')
        self.root.update_idletasks() # Force UI update

        try:
            prep_img = self.crop_and_preprocess(self.original_cv_image)
            prediction_score = self.model.predict(prep_img, verbose=0)[0][0]
            self.update_results(prediction_score)
        except Exception as e:
            messagebox.showerror("Processing Error", f"Model failed to predict: {e}")

    def update_results(self, score):
        has_tumor = score > 0.5
        confidence = score if has_tumor else (1 - score)

        res_text = "⚠️ TUMOR DETECTED" if has_tumor else "✅ NO TUMOR DETECTED"
        res_color = "#e74c3c" if has_tumor else "#27ae60"

        self.prediction_label.config(text=res_text, fg=res_color)
        self.confidence_text.config(text=f"{confidence*100:.2f}%", fg=res_color)
```

GUI INTERFACE LAYOUT AND USER WORKFLOW DESIGN

```
def setup_ui(self):
    # Header
    header_frame = tk.Frame(self.root, bg='#2c3e50', height=80)
    header_frame.pack(fill=tk.X)
    header_frame.pack_propagate(False)

    title_label = tk.Label(header_frame, text="🧠 Brain Tumor Detection & Advisory System",
                           font=("Arial", 22, "bold"), bg='#2c3e50', fg='white')
    title_label.pack(pady=20)

    # Main content
    content_frame = tk.Frame(self.root, bg='#f0f0f0')
    content_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

    # Left Panel (Image)
    left_frame = tk.Frame(content_frame, bg='white', relief=tk.RIDGE, bd=2)
    left_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0, 10))

    tk.Label(left_frame, text="MRI Image Preview", font=("Arial", 14, "bold"), bg='white', fg='#2c3e50').pack(pady=10)

    self.image_canvas = tk.Canvas(left_frame, width=450, height=450, bg='#ecf0f1', highlightthickness=0)
    self.image_canvas.pack(pady=10)
    self.image_canvas.create_text(225, 225, text="Step 1: Upload MRI\nStep 2: Click Analyze",
                                  font=("Arial", 12), fill='#95a5a6', justify=tk.CENTER, tags="placeholder")
```

- **The system features a clean, medical-grade interface with a header, image preview panel, and results display. The canvas shows uploaded MRI scans with intuitive prompts, creating a user-friendly workflow from image upload to diagnostic analysis in a clinically appropriate visual format.**

INTERACTIVE CONTROLS AND DIAGNOSTIC OUTPUT PANEL

```
# Buttons Container
btn_frame = tk.Frame(left_frame, bg='white')
btn_frame.pack(fill=tk.X, padx=20, pady=10)

self.upload_btn = tk.Button(btn_frame, text="📁 Upload MRI Image", font=("Arial", 12, "bold"),
                            bg='#3498db', fg='white', cursor='hand2', command=self.upload_image)
self.upload_btn.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5)

self.analyze_btn = tk.Button(btn_frame, text="⚡ Analyze Image", font=("Arial", 12, "bold"),
                             bg='#e67e22', fg='white', cursor='hand2', state=tk.DISABLED, command=self.run_detection)
self.analyze_btn.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5)

# Right Panel (Results)
right_frame = tk.Frame(content_frame, bg='#f0f0f0')
right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

results_frame = tk.Frame(right_frame, bg='white', relief=tk.RIDGE, bd=2)
results_frame.pack(fill=tk.X, pady=(0, 10))

tk.Label(results_frame, text="AI Diagnostic Output", font=("Arial", 15, "bold"), bg='white', fg='#2c3e50').pack(pady=10)

self.prediction_label = tk.Label(results_frame, text="Awaiting Analysis...", font=("Arial", 18, "bold"), bg='white', fg='#7f8c8d')
self.prediction_label.pack(pady=5)

self.confidence_text = tk.Label(results_frame, text="---%", font=("Arial", 24, "bold"), bg='white', fg='#2c3e50')
self.confidence_text.pack()

self.progress_canvas = tk.Canvas(results_frame, width=280, height=20, bg='#ecf0f1', highlightthickness=0)
self.progress_canvas.pack(pady=10)

self.timestamp_label = tk.Label(results_frame, text="", font=("Arial", 9), bg='white', fg='#7f8c8d')
self.timestamp_label.pack(pady=5)
```

- **The interface features sequential workflow buttons—first upload MRI, then analyze—with visual feedback (color-coded states) to guide users. Results display includes confidence percentages, progress visualization, and timestamps, creating a structured diagnostic process familiar to medical professionals.**

MEDICAL ADVISORY PANEL AND IMAGE PROCESSING IMPLEMENTATION

```
    # Recommendations
    rec_frame = tk.Frame(right_frame, bg='white', relief=tk.RIDGE, bd=2)
    rec_frame.pack(fill=tk.BOTH, expand=True)
    tk.Label(rec_frame, text="Full Medical Advisory", font=("Arial", 14, "bold"), bg='white').pack(pady=10)

    self.recommendations_text = scrolledtext.ScrolledText(rec_frame, wrap=tk.WORD, font=("Arial", 10), bg='#f8f9fa', state=tk.DISABLED)
    self.recommendations_text.pack(fill=tk.BOTH, expand=True, padx=15, pady=(0, 15))

    tk.Button(right_frame, text="🗑️ Reset System", bg='#95a5a6', fg='white', command=self.clear_results).pack(pady=10, fill=tk.X)

def crop_and_preprocess(self, image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) > 0:
        c = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(c)
        cropped = image[y:y+h, x:x+w]
    else:
        cropped = image

    resized = cv2.resize(cropped, self.IMG_SIZE, interpolation=cv2.INTER_CUBIC)
    img_array = np.expand_dims(resized, axis=0)
    return vgg_preprocess(img_array.astype('float32'))
```

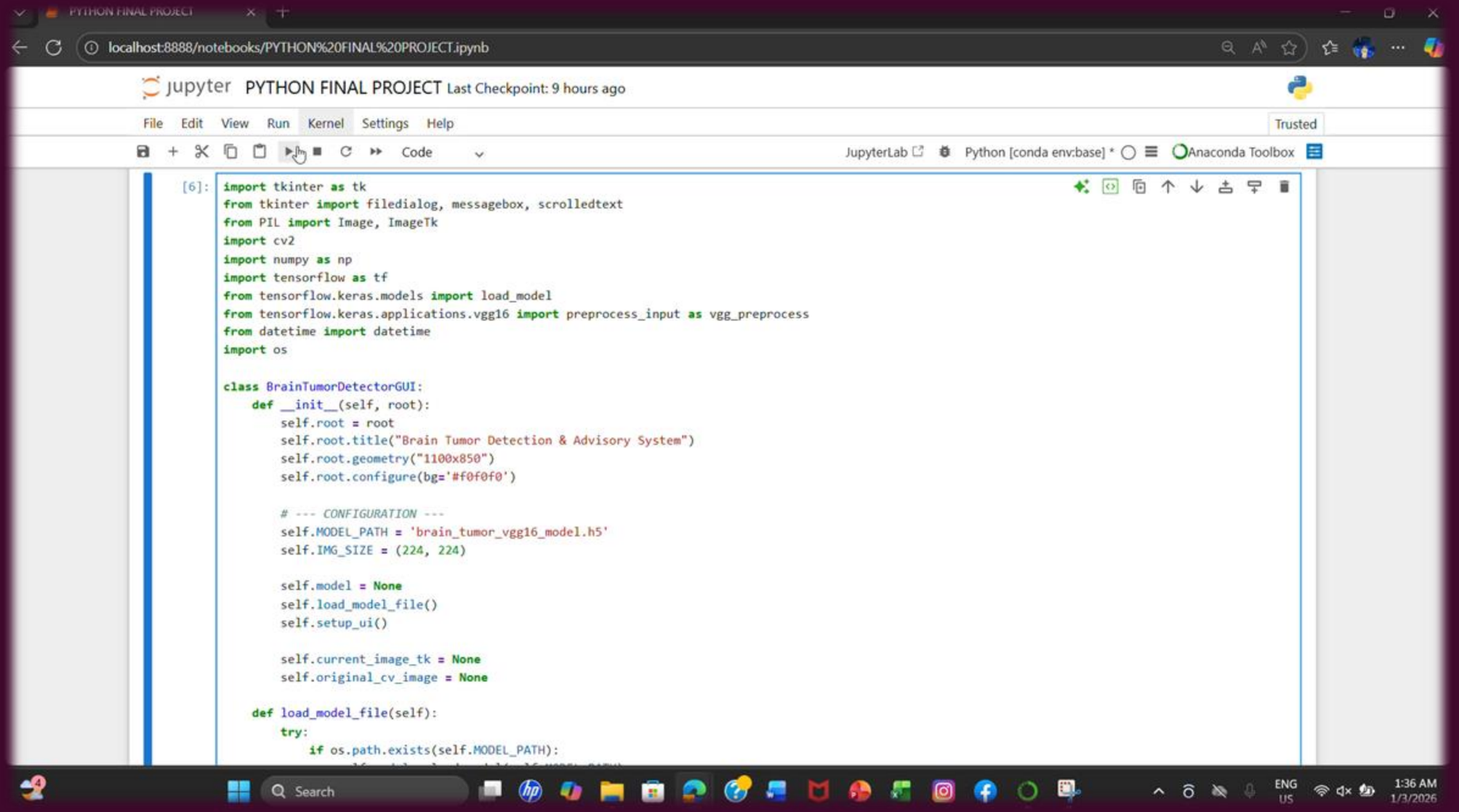
The GUI features a scrollable medical recommendations section providing detailed, diagnosis-specific guidance following each analysis. The integrated image processing pipeline applies the same brain-cropping algorithm used during model training, ensuring diagnostic consistency between development and clinical deployment.

SYSTEM RESET FUNCTION AND APPLICATION ENTRY POINT

```
    def clear_results(self):
        self.image_canvas.delete("all")
        self.image_canvas.create_text(225, 225, text="No image loaded", fill='#95a5a6')
        self.prediction_label.config(text="Awaiting Analysis...", fg='#7f8c8d')
        self.confidence_text.config(text="---%", fg='#2c3e50')
        self.progress_canvas.delete("bar")
        self.recommendations_text.config(state=tk.NORMAL)
        self.recommendations_text.delete(1.0, tk.END)
        self.recommendations_text.config(state=tk.DISABLED)
        self.analyze_btn.config(state=tk.DISABLED, bg='#95a5a6')
        self.original_cv_image = None

if __name__ == "__main__":
    root = tk.Tk()
    app = BrainTumorDetectorGUI(root)
    root.mainloop()
```

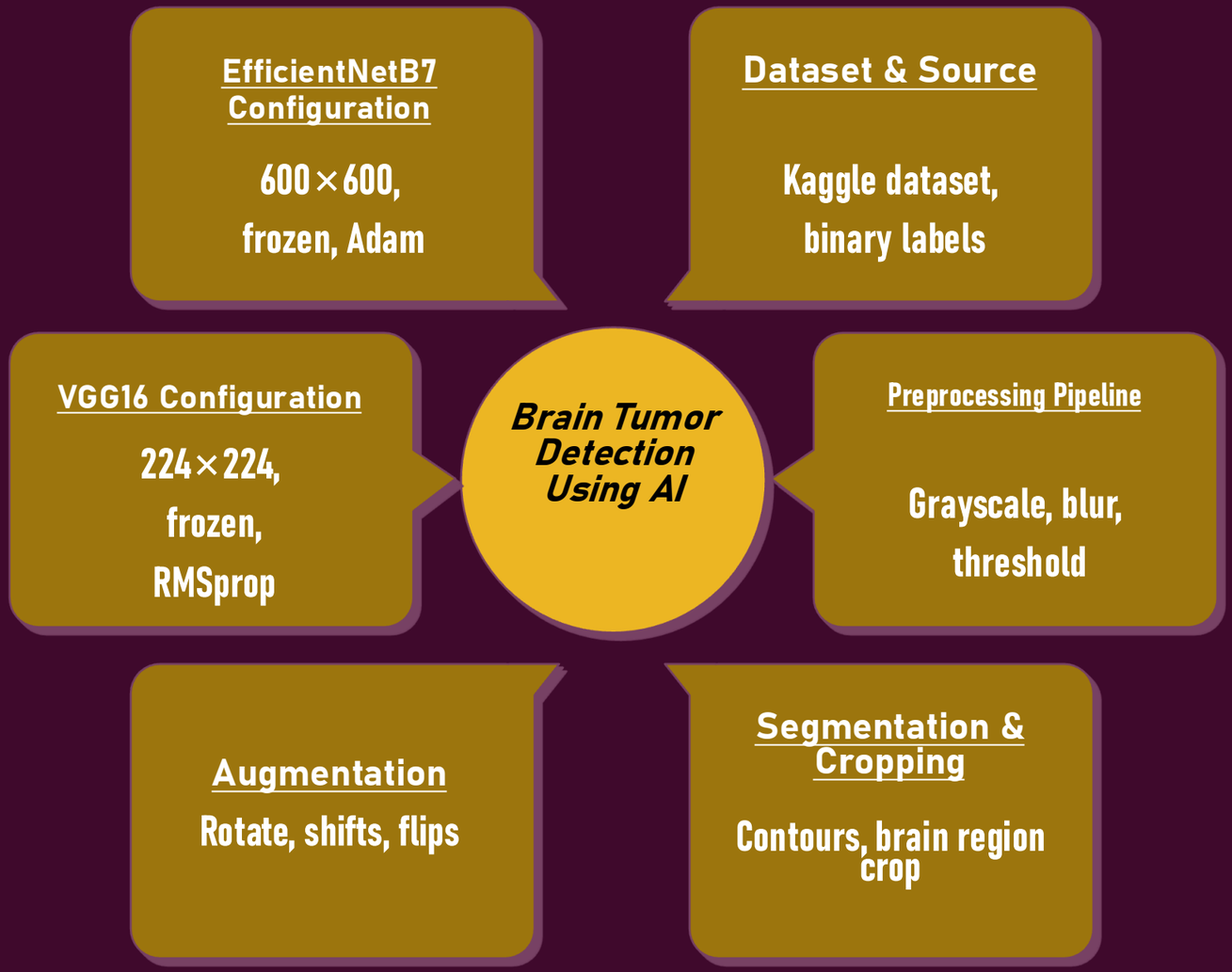
A comprehensive reset function clears all patient data, diagnostic displays, and image previews, preparing the system for new cases while maintaining clinical privacy standards and enabling efficient multi-patient workflows in medical environments.

SAMPLE VIDEO FOR TESTING USER INTERFACE RESULTS USING MRI IMAGES



SO THIS IS HOW WE TEST THE MODEL USING THE GUI, WHERE WE CAN PUT MRI IMAGES OF THE BRAIN AND SEE WHETHER IT CONTAINS A BRAIN TUMOR OR NOT BY ANALYZING THE MODEL. THE CONFIDENCE FOR BOTH SITUATIONS IS GOOD ENOUGH.

****SUMMARY OF OUR CODE****



CONCLUSION

- An AI-based Brain Tumor Detection System was successfully developed using VG16 and EfficientNetB7.
- VGG16 was chosen for deployment due to its good accuracy and efficiency.
- The system accurately classifies MRI images into Tumor and No Tumor categories.
- A GUI based application allows easy image upload, prediction, confidence display, and medical advisory output.
- Artificial Intelligence has strong potential to assist radiologists by improving early detection, reducing diagnostic time, and supporting better clinical decisions.

THANK YOU
THE END