
image2(37).jpg

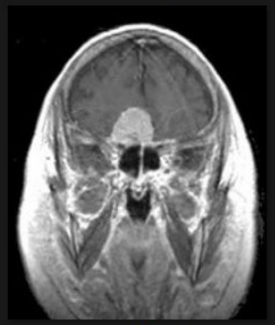
image2(38).jpg

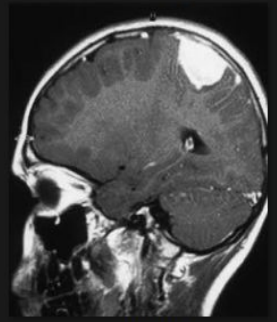
image2(39).jpg

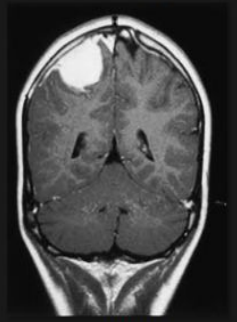
image2(40).jpg

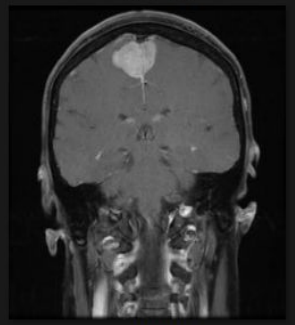
image2(41).jpg

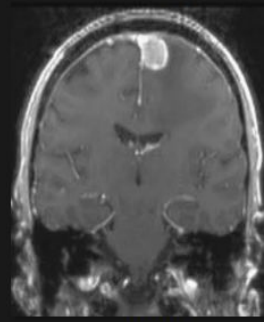
image2(42).jpg

CNN Brain Tumor Predictor Model

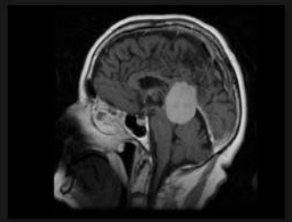
image2(49).jpg

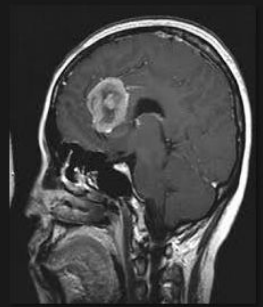
image2(50).jpg

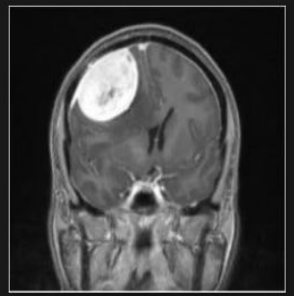
image2(51).jpg

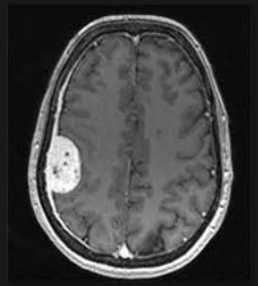
image2(52).jpg

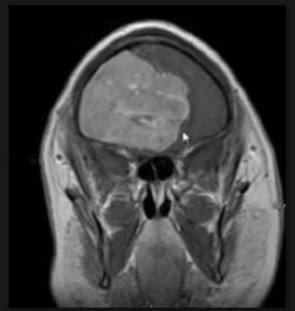
image2(53).jpg


image2(54).jpg


image2(37).jpg

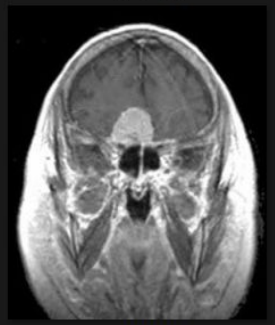
image2(38).jpg

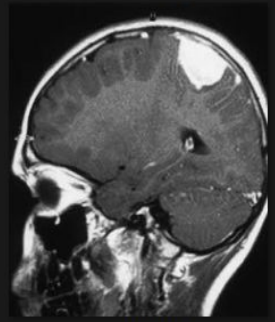
image2(39).jpg

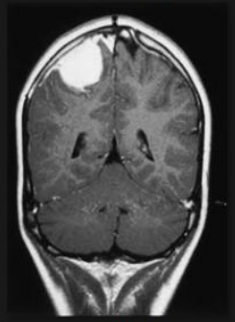
image2(40).jpg

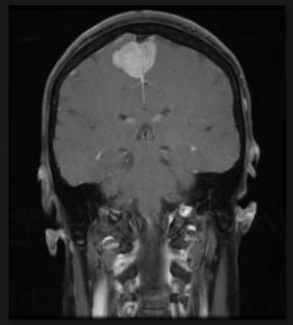
image2(41).jpg

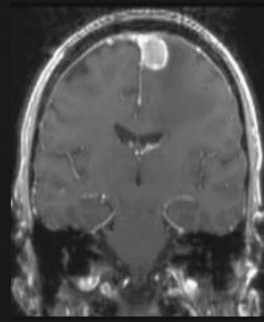
image2(42).jpg

Objective: Correctly classify brain tumors

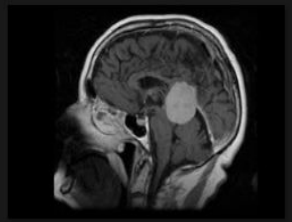
image2(49).jpg

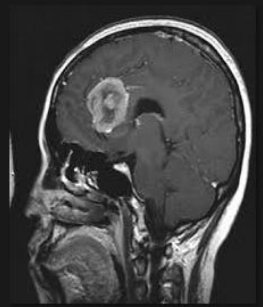
image2(50).jpg

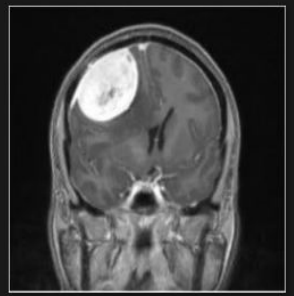
image2(51).jpg

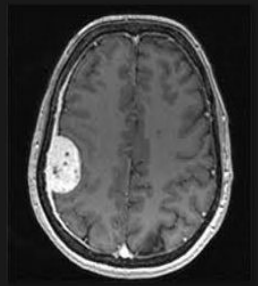
image2(52).jpg

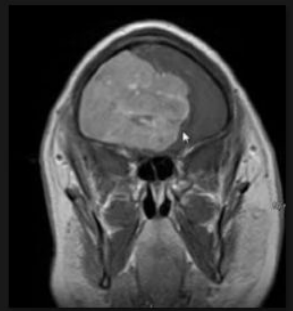
image2(53).jpg


image2(54).jpg

Preprocessing A

- Image Augmentation
- Balancing the dataset

```python
# Determine the number of augmented images
augmentations_per_image = 6  # 6 * 395 = 2370 augmented images

# List all 'no_tumor' images
no_tumor_images = os.listdir(no_tumor_dir)

# Process and augment each 'no_tumor' image
for img_name in no_tumor_images:
    # Load the image
    img_path = os.path.join(no_tumor_dir, img_name)
    img = load_img(img_path)
    x = img_to_array(img)
    x = x.reshape((1,) + x.shape)  # Reshape

    # Initialize a counter for this image
    i = 0
    # Generate augmented images
    for batch in augmentation.flow(x, batch_size=1, save_to_dir=save_dir, save_prefix='no_tumor', save_format='jpg'):
        i += 1
        if i >= augmentations_per_image:
            break  # Stop the loop once we hit the target
```

Preprocessing B

- Batch size
- Determining class mode
- Separating validation data
- Rescaling and shape

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

image_width, image_height = 224, 224

# Create an ImageDataGenerator for training with data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalize pixel values
    horizontal_flip=True,  # Randomly flip inputs horizontally
    validation_split=0.2)  # validation split

# Flow training images in batches using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        '../MRI_DATA/Training/',
        target_size=(image_width, image_height),
        batch_size=32,
        class_mode='binary',
        subset='training')  # Set as training data

# Flow validation images in batches using train_datagen generator
validation_generator = train_datagen.flow_from_directory(
        '../MRI_DATA/Training/',
        target_size=(image_width, image_height),
        batch_size=32,
        class_mode='binary',
        subset='validation')
```

✓ 1.6s

Constructing the Model

- 4 convolutional layers
- Flattening
- Activation: Sigmoid vs softmax

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    # Convolutional layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_width, image_height, 3)),
    MaxPooling2D(2, 2),

    # Convolutional layer 2
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    # Convolutional layer 3
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    # Convolutional layer 4
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    # Flattening the 3D output to 1D
    Flatten(),

    # Dropout to reduce overfitting
    Dropout(0.5),

    # Dense layer for prediction
    Dense(512, activation='relu'),

    # Output layer
    Dense(1, activation='sigmoid')
])

# Printing the model summary to review the architecture
model.summary()
```

Training the model

- Had some long training sessions
- 22 training sessions total

```
# training the model

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=150,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size)
```

920m 43.5s

```
Epoch 1/150
71/71 [==============================] - 313s 4s/step - loss: 1.4560 - accuracy: 0.4525 - val_loss: 1.2693 - val_accuracy: 0.3254
Epoch 2/150
71/71 [==============================] - 290s 4s/step - loss: 0.9302 - accuracy: 0.5943 - val_loss: 1.2672 - val_accuracy: 0.4081
Epoch 3/150
71/71 [==============================] - 294s 4s/step - loss: 0.7293 - accuracy: 0.6883 - val_loss: 1.0880 - val_accuracy: 0.5662
Epoch 4/150
71/71 [==============================] - 294s 4s/step - loss: 0.6343 - accuracy: 0.7333 - val_loss: 1.0177 - val_accuracy: 0.5221
Epoch 5/150
71/71 [==============================] - 293s 4s/step - loss: 0.5792 - accuracy: 0.7585 - val_loss: 1.0411 - val_accuracy: 0.5570
Epoch 6/150
71/71 [==============================] - 294s 4s/step - loss: 0.4959 - accuracy: 0.7978 - val_loss: 1.0894 - val_accuracy: 0.5846
Epoch 7/150
71/71 [==============================] - 291s 4s/step - loss: 0.5037 - accuracy: 0.7951 - val_loss: 0.9968 - val_accuracy: 0.6268
Epoch 8/150
71/71 [==============================] - 291s 4s/step - loss: 0.4596 - accuracy: 0.8137 - val_loss: 0.8350 - val_accuracy: 0.6544
Epoch 9/150
71/71 [==============================] - 292s 4s/step - loss: 0.4147 - accuracy: 0.8366 - val_loss: 1.0226 - val_accuracy: 0.6379
```

Optimizing the Model & Hypertuning

- Keras tuner
- Manual Hypertuning
- Comparing Results

```python
# Define the function to build the model 2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

def build_model(hp):
    model = Sequential()
    model.add(Conv2D(hp.Int('conv1_units', min_value=32, max_value=256, step=32), (3, 3), activation=hp.Choice('activation', ['relu', 'sigmoid', 'tanh', 'elu', 'selu']), i
    model.add(MaxPooling2D(2, 2))

    model.add(Conv2D(hp.Int('conv2_units', min_value=32, max_value=256, step=32), (3, 3), activation=hp.Choice('activation', ['relu', 'sigmoid', 'tanh', 'elu', 'selu'])))
    model.add(MaxPooling2D(2, 2))

    model.add(Conv2D(hp.Int('conv3_units', min_value=32, max_value=256, step=32), (3, 3), activation=hp.Choice('activation', ['relu', 'sigmoid', 'tanh', 'elu', 'selu'])))
    model.add(MaxPooling2D(2, 2))

    model.add(Conv2D(hp.Int('conv4_units', min_value=32, max_value=256, step=32), (3, 3), activation=hp.Choice('activation', ['relu', 'sigmoid', 'tanh', 'elu', 'selu'])))
    model.add(MaxPooling2D(2, 2))

    model.add(Flatten())

    model.add(Dropout(hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1)))

    model.add(Dense(hp.Int('dense_units', min_value=128, max_value=1024, step=128), activation=hp.Choice('activation', ['relu', 'sigmoid', 'tanh', 'elu', 'selu'])))
    model.add(Dense(4, activation='softmax'))

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience = 25, restore_best_weights=True)
```


image2(37).jpg

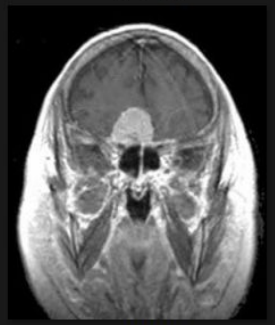
image2(38).jpg

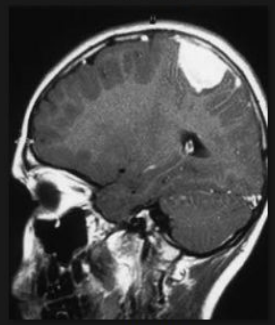
image2(39).jpg

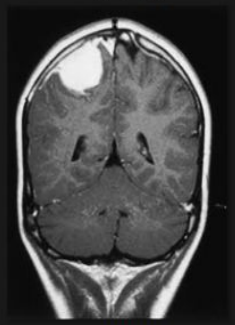
image2(40).jpg

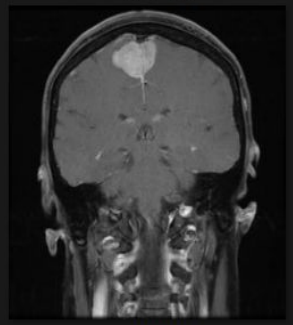
image2(41).jpg

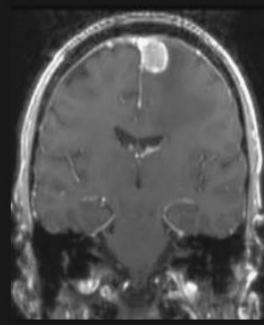
image2(42).jpg

Results and Demo

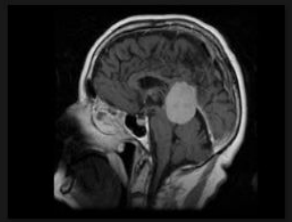
image2(49).jpg

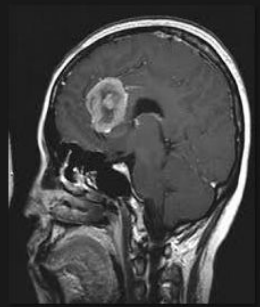
image2(50).jpg

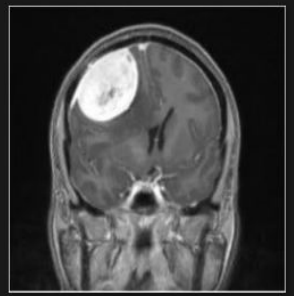
image2(51).jpg

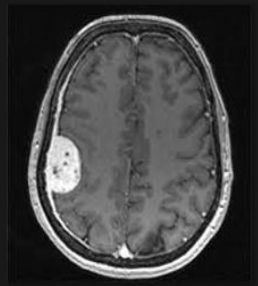
image2(52).jpg

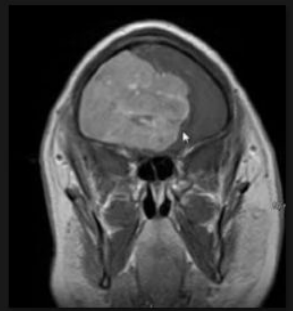
image2(53).jpg


image2(54).jpg