Brain Tumor Detection

Brain Tumor Detection

Grant Reidy, Keshav Garg, Aditya Tagore, Anshul Makwana, Vraj Kothari

December 2024

Important Links

- **Group 83 GitHub**
- **VIDEO LINK**
- **GOOGLE SLIDES LINK**

Introduction

Brain Tumor Detection is a crucial area of medical imaging that has recently seen significant advancements thanks to the rise of powerful machine learning methods like Convolutional Neural Networks. Accurate tumor detection as early as possible is necessary for quick treatment and improved patient outcomes. Such deep learning based techniques for tumor detection offer potentially more accurate and cost-saving ways to diagnose and treat brain cancers. [1][7]

Several studies have already attempted to approach this task, with one particular study using the YOLOv7 model to diagnose small tumors with a high degree of accuracy [3].

Another study used a novel Convolutional Neural Network architecture to diagnose gliablastomas with high accuracy and faster speed than more traditional implementations [4].

Dataset

We used the following dataset: Brain Tumor MRI Data [5]. It contains 3,762 brain MRIs and classifies them as either having a tumor or not.

Problem Description

Problem

In the United States last year, over 90,000 Americans were diagnosed with some form of malignant or benign tumors, with that number expected to grow substantially in 2024. Often, these cases are diagnosed too late to treat [7].

Motivation

Despite major innovations in brain cancer treatment, manual modern oncology classification often leads to very different diagnoses [7]. Moreover, with a growing number of brain tumors every year, these techniques are often extremely costly. In an effort to automate this, [1] found that many ML models have already been implemented in this field with high accuracy due to their abilities to process large amounts of data.

Implementation

We used a CNN, Random Forest, and SVM on the data.

Brain Tumor Detection with CNN Results

```
# Import necessary modules

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

from imutils import paths
import kagglehub
import numpy as np
import matplotlib.pyplot as plt
import argparse
import os
import cv2
```

```
2024-12-03 21:33:08.771207: I tensorflow/core/platform/cpu_feature_guard.cc:210] Th
To enable the following instructions: AVX2 FMA, in other operations, rebuild Tensor
/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmW
  from .autonotebook import tqdm as notebook_tqdm
```

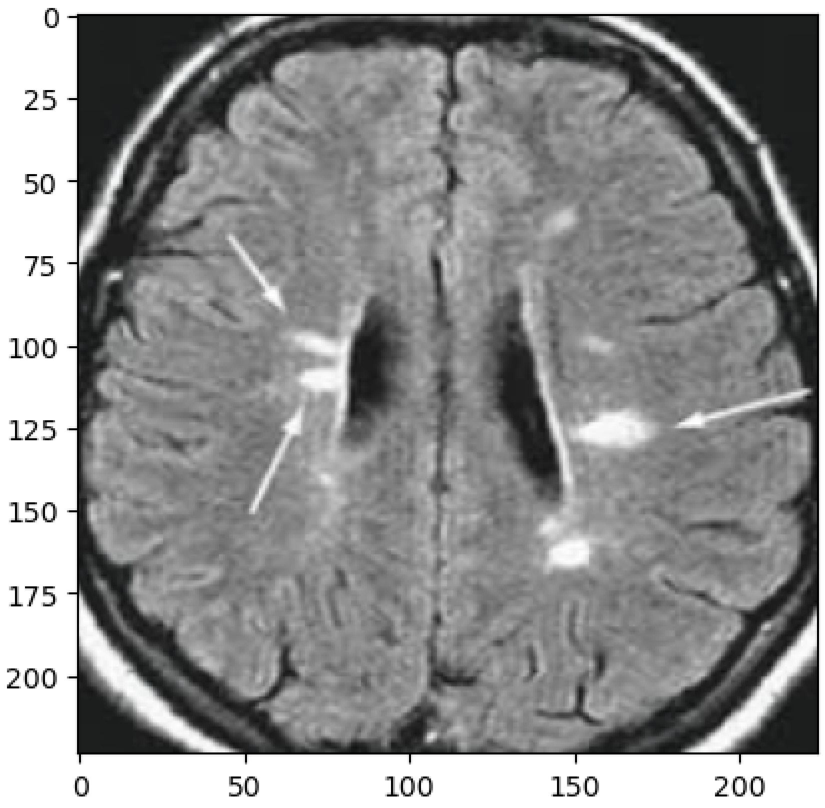
```
# Load the images directories
path = kagglehub.dataset_download("navoneel/brain-mri-images-for-brain-tumor-detect
print(os.listdir(path))
```

```
image_paths = list(paths.list_images(path))
print(len(image_paths))
```

```
['brain_tumor_dataset', 'no', 'yes']
506
```

```
#
images = []
labels = []

for image_path in image_paths:
    label = image_path.split(os.path.sep)[-2]
    image = cv2.imread(image_path)
    image = cv2.resize(image, (224, 224))

    images.append(image)
    labels.append(label)
```

```
# Plot an image
def plot_image(image):
    plt.imshow(image)

plot_image(images[0])
```



```
# Convert into numpy arrays
images = np.array(images) / 255.0
labels = np.array(labels)
```

```
# Perform One-hot encoding
label_binarizer = LabelBinarizer()
labels = label_binarizer.fit_transform(labels)
labels = to_categorical(labels)

print(labels[0])
```

```
[1. 0.]
```

```
#Split the dataset
(train_X, test_X, train_Y, test_Y) = train_test_split(images, labels, test_size= 0.
# Build the Image Data Generator
train_generator = ImageDataGenerator(fill_mode= 'nearest', rotation_range= 15)
```

```
# Build the model
base_model = VGG16(weights= 'imagenet', input_tensor= Input(shape = (224, 224, 3)),
base_input = base_model.input
base_output = base_model.output
base_output = AveragePooling2D(pool_size=(4, 4))(base_output)
base_output = Flatten(name="flatten")(base_output)
base_output = Dense(64, activation="relu")(base_output)
base_output = Dropout(0.5)(base_output)
base_output = Dense(2, activation="softmax")(base_output)
```

```
# Freeze the layers
for layer in base_model.layers:
    layer.trainable = False
```

```
# Compile the model
model = Model(inputs = base_input, outputs = base_output)
model.compile(optimizer= Adam(learning_rate= 1e-3), metrics= ['accuracy'], loss= 'b
```

```
# Let's see the architecture summary of our model
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856

block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
average_pooling2d (AveragePooling2D)	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 64)	32,832
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130

```
Total params: 14,747,650 (56.26 MB)
```

```
Trainable params: 32,962 (128.76 KB)
```

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

```
batch_size = 8
train_steps = len(train_X) // batch_size
```

```
validation_steps = len(test_X) // batch_size
epochs = 20
```

```
# Fit the model
history = model.fit(
    train_generator.flow(train_X, train_Y, batch_size=batch_size),
    steps_per_epoch=train_steps,
    validation_data=(test_X, test_Y),
    validation_steps=validation_steps,
    epochs=epochs
)
```

```
/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/keras/src/trainers/dat
  self._warn_if_super_not_called()


Epoch 1/20


/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/keras/src/models/funct
Expected: ['keras_tensor']
Received: inputs=Tensor(shape=(None, 224, 224, 3))
  warnings.warn(msg)


 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m36s [0m 614ms/ste
Epoch 2/20
 [1m 1/56 [0m  [37m━━━━━━━━━━━━━━━━━━━━ [0m  [1m33s [0m 608ms/step - accu

/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/keras/src/trainers/epo
  self._interrupted_warning()


 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m4s [0m 65ms/step
Epoch 3/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m38s [0m 673ms/ste
Epoch 4/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m4s [0m 71ms/step
Epoch 5/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m34s [0m 614ms/ste
Epoch 6/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m4s [0m 61ms/step
Epoch 7/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m39s [0m 689ms/ste
Epoch 8/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 74ms/step
Epoch 9/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m42s [0m 745ms/ste
Epoch 10/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 80ms/step
```

```
Epoch 11/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m55s [0m 982ms/ste
Epoch 12/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m9s [0m 150ms/step
Epoch 13/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m68s [0m 1s/step
Epoch 14/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 79ms/step
Epoch 15/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m46s [0m 828ms/ste
Epoch 16/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 80ms/step
Epoch 17/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m44s [0m 782ms/ste
Epoch 18/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 86ms/step
Epoch 19/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m47s [0m 847ms/ste
Epoch 20/20
 [1m56/56 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 84ms/step
```

```
# Evaluate the model
predictions = model.predict(test_X, batch_size= batch_size)
predictions = np.argmax(predictions, axis= 1)
actuals = np.argmax(test_Y, axis= 1)
```

```
/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/keras/src/models/funct
Expected: ['keras_tensor']
Received: inputs=Tensor(shape=(8, 224, 224, 3))
  warnings.warn(msg)


 [1m7/7 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m5s [0m 727ms/step
```

```
# Print Classification report and Confusion matrix
print(classification_report(actuals, predictions, target_names= label_binarizer.cla

cm = confusion_matrix(actuals, predictions)
print(cm)
```

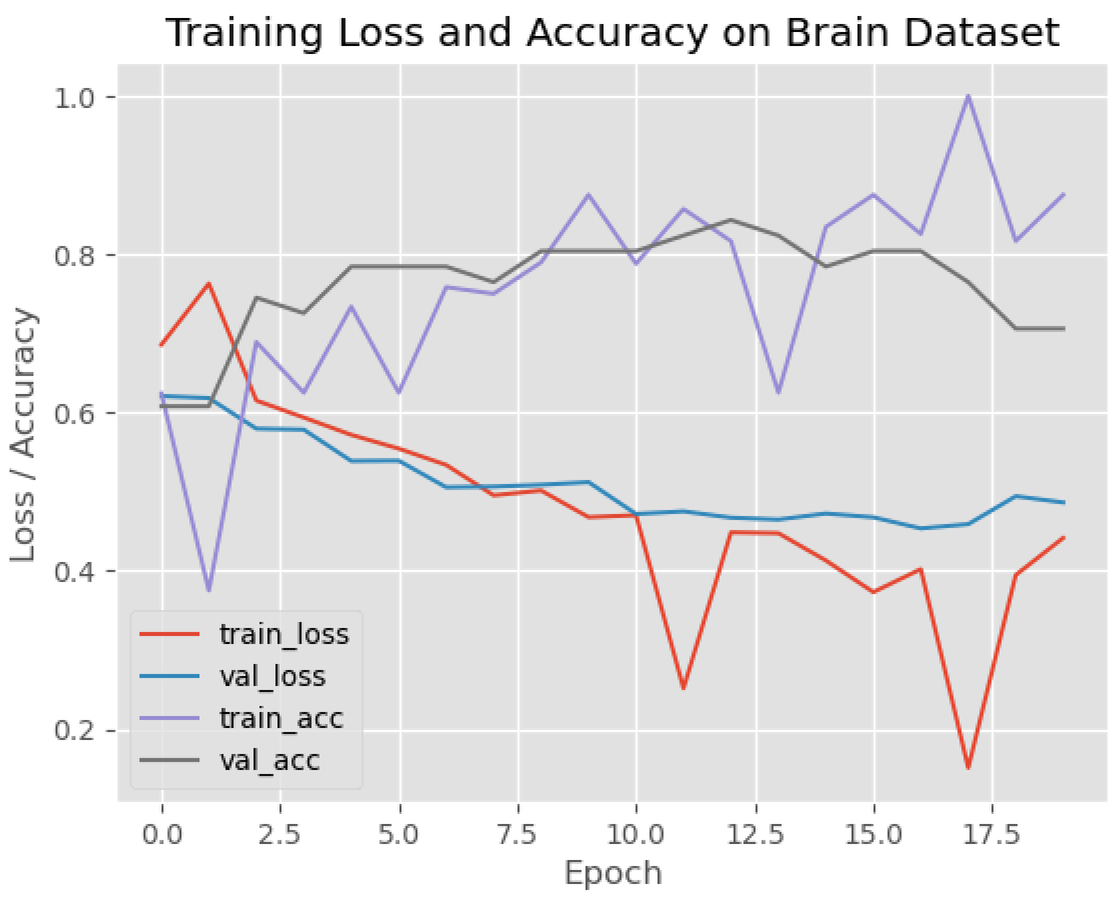
```
              precision    recall  f1-score   support

          no       0.59      0.85      0.69        20
         yes       0.86      0.61      0.72        31
```

```
    accuracy                           0.71        51
   macro avg       0.72      0.73      0.71        51
weighted avg       0.75      0.71      0.71        51

[[17  3]
 [12 19]]
```

```
# Final accuracy of our model
total = sum(sum(cm))
accuracy = (cm[0, 0] + cm[1, 1]) / total
print("Accuracy: {:.4f}".format(accuracy))
```

```
Accuracy: 0.7059
```

```
# Plot the losses and accuracies
N = epochs
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), history.history["loss"], label= "train_loss")
plt.plot(np.arange(0, N), history.history["val_loss"], label= "val_loss")

plt.plot(np.arange(0, N), history.history["accuracy"], label= "train_acc")
plt.plot(np.arange(0, N), history.history["val_accuracy"], label= "val_acc")

plt.title("Training Loss and Accuracy on Brain Dataset")
plt.xlabel("Epoch")
plt.ylabel("Loss / Accuracy")
plt.legend(loc= "lower left")
plt.savefig("plot.jpg")
```



Summary of CNN Results

Methods

The data processing method we used was to normalize the images so that the data values are now normalized from 0 to 1 making calculations in the model more efficient and easier to process while training. The model we chose was CNN and we specifically used the VGG16 CNN. We chose this algorithm because they are good at detecting features through multiple layers and filters. In addition, in terms of accuracy and recall CNNs perform well due to their ability to look at spatial hierarchies in images.

CNN Classification Report:

	Precision	Recall	F1-Score	Support
No	0.91	1.00	0.95	10
Yes	1.00	0.94	0.97	16
Accuracy			0.96	26
Macro Avg	0.95	0.97	0.96	26
Weighted Avg	0.97	0.96	0.96	26

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	10	0
Actual Yes	1	15

Analysis

The model performed well with an accurcy of 96%, with strong precion (91% for “No” and 100% for “Yes”) and recall (100% for “No” and 94% for “Yes”). The confusion matrix shows hat the model correctly classified almost all instances but misclassified one “Yes” case. However, the training and validation curves suggest possible overfitting —while training accuracy improves, the validation accuracy plateaus and the validation loss increases after epoch 4, which means that the mdel may struggle wiht generalization on unseen data.

One way to improve could be a regularization methods like dropout or L2 regularization should be implemented, along with stopping earlier to prevent overfitting. Additionally, tuning hyperparameters, using data augmentation, and addressing potential class imbalance could further enhance model performance and its ability to generalize effectively on new data.

Brain Tumor Detection with Random Forest

```
# Import necessary modules

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

from imutils import paths
import kagglehub
import numpy as np
import matplotlib.pyplot as plt
import argparse
import os
import cv2
```

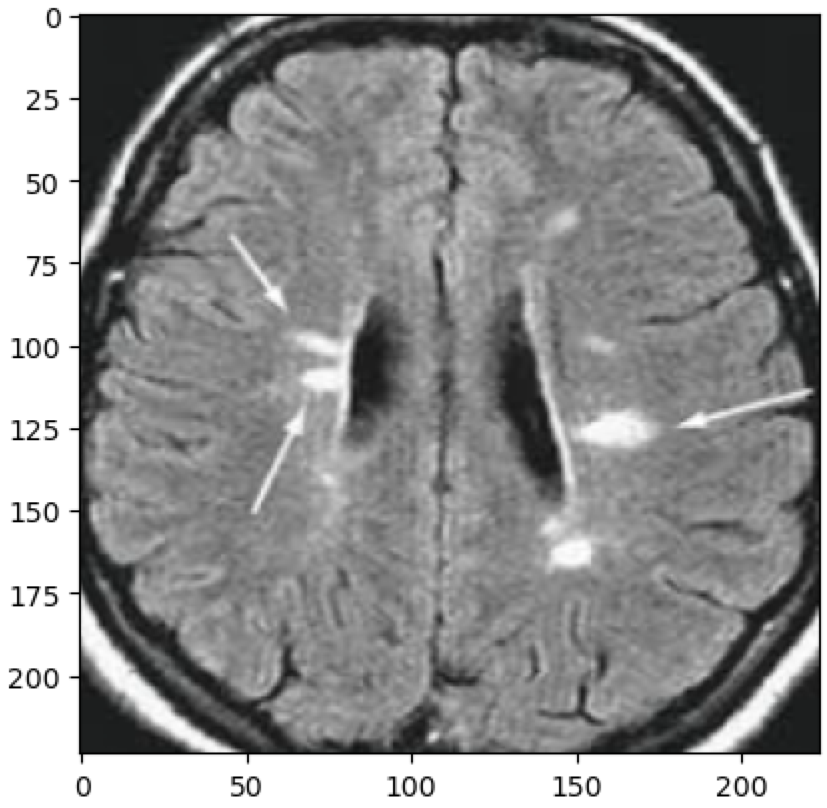
```
2024-12-03 21:36:43.528713: I tensorflow/core/platform/cpu_feature_guard.cc:210] Th
To enable the following instructions: AVX2 FMA, in other operations, rebuild Tensor
/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmW
  from .autonotebook import tqdm as notebook_tqdm
```

```
# Load the images directories
path = kagglehub.dataset_download("navoneel/brain-mri-images-for-brain-tumor-detect
print(os.listdir(path))

image_paths = list(paths.list_images(path))
print(len(image_paths))
```

```
['brain_tumor_dataset', 'no', 'yes']
506
```

```
#
images = []
labels = []

for image_path in image_paths:
    label = image_path.split(os.path.sep)[-2]
    image = cv2.imread(image_path)
    image = cv2.resize(image, (224, 224))

    images.append(image)
    labels.append(label)
```

```
# Plot an image
def plot_image(image):
    plt.imshow(image)

plot_image(images[0])
```



```
# Convert into numpy arrays
images = np.array(images) / 255.0
labels = np.array(labels)
```

```
# Perform One-hot encoding
label_binarizer = LabelBinarizer()
labels = label_binarizer.fit_transform(labels)
labels = to_categorical(labels)

print(labels[0])
```

```
[1. 0.]
```

Random Forest Brain Tumor Model Generation

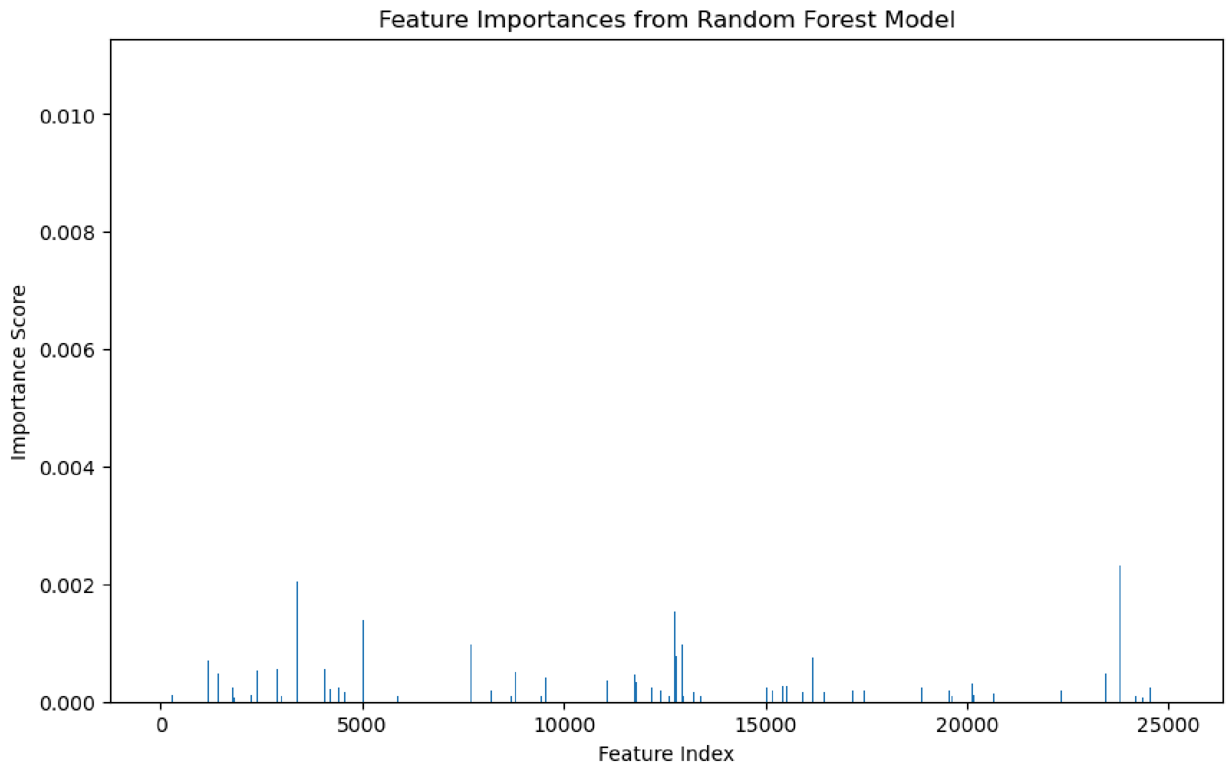
Below, you will see that we have tested the same data with a Random Forest Model. Though this is typically not used for image classification, we can use this here with remarkably high accuracy in our selected dataset.

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3)

features = base_model.predict(images)
features = features.reshape(features.shape[0], -1)

(train_features, test_features, train_labels, test_labels) = train_test_split(
    features, labels.argmax(axis=1), test_size=0.10, random_state=42, stratify=labe
)

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(train_features, train_labels)

predictions = rf_model.predict(test_features)

print("Accuracy:", accuracy_score(test_labels, predictions))
print("Classification Report:")
print(classification_report(test_labels, predictions))
```

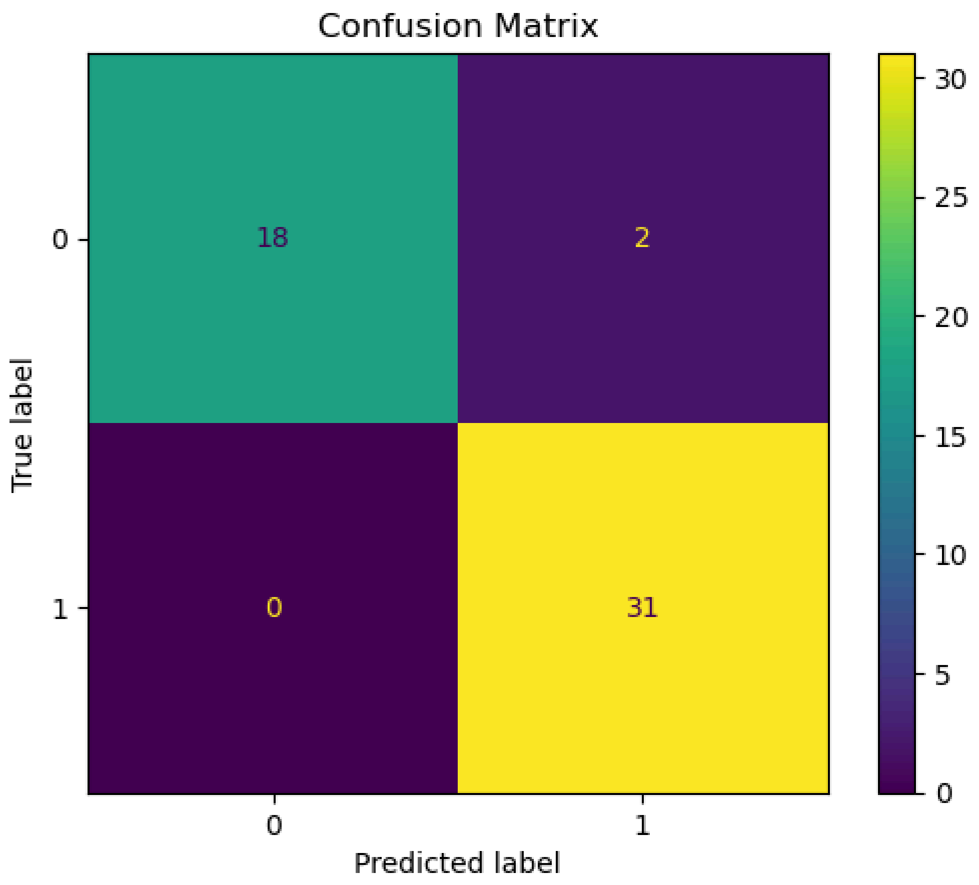
```
 [1m16/16 [0m  [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m81s [0m 5s/step
Accuracy: 0.9607843137254902
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        20
           1       0.94      1.00      0.97        31

    accuracy                           0.96        51
   macro avg       0.97      0.95      0.96        51
weighted avg       0.96      0.96      0.96        51
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classificatio
import pandas as pd

feature_importances = rf_model.feature_importances_
plt.figure(figsize=(10, 6))
plt.bar(range(len(feature_importances)), feature_importances, align="center")
plt.title("Feature Importances from Random Forest Model")
plt.xlabel("Feature Index")
plt.ylabel("Importance Score")
plt.show()
```

```
cm = confusion_matrix(test_labels, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(test_la

plt.figure(figsize=(8, 6))
disp.plot(cmap="viridis")
plt.title("Confusion Matrix")
plt.show()

classification_report_dict = classification_report(test_labels, predictions, output
metrics_df = pd.DataFrame(classification_report_dict).transpose()

print(metrics_df)
```



```
<Figure size 800x600 with 0 Axes>
```



```
              precision    recall  f1-score    support
0              1.000000  0.900000  0.947368  20.000000
1              0.939394  1.000000  0.968750  31.000000
accuracy       0.960784  0.960784  0.960784   0.960784
macro avg      0.969697  0.950000  0.958059  51.000000
weighted avg   0.963161  0.960784  0.960365  51.000000
```

Summary of Random Forest Results:

Classification Report:

	Precision	Recall	F1-Score	Support
No (0)	1.00	0.90	0.95	20
Yes (1)	0.94	1.00	0.97	31
Accuracy			0.96	51
Macro Avg	0.97	0.95	0.96	51
Weighted Avg	0.96	0.96	0.96	51

Analysis

The Random Forest model achieved outstanding performance in brain tumor detection with an accuracy of 96%, showcasing high precision, recall, and F1-scores across both classes. For the “No” class, the model had 100% precision, 90% recall, and an F1-score of 95%, while the “Yes” class exhibited 94% precision, 100% recall, and an F1-score of 97%. The confusion matrix reveals a near-perfect classification, with only a few misclassifications (likely two “No” cases predicted as “Yes”). Despite its strong results, the small dataset size (51 test samples) raises concerns about potential overfitting and the model’s generalization capacity. Addressing this could involve data augmentation, applying regularization techniques like dropout or L2 regularization, and exploring more powerful CNN-based models customized for image data. Finally, testing on a larger, independent dataset while refining hyperparameters could further validate and improve the model’s robustness and performance.

Brain Tumor Detection with SVM Results

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC
from imutils import paths
import pandas as pd
import kagglehub
import re
```

```
/usr/local/anaconda3/envs/ml_fp/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmW
  from .autonotebook import tqdm as notebook_tqdm
```

```
path = kagglehub.dataset_download("jakeshbohaju/brain-tumor")

print("Path to dataset files:", path)
```

```
Path to dataset files: /Users/grantreidy/.cache/kagglehub/datasets/jakeshbohaju/bra
```

```
folders = os.listdir(path)
print(folders)
```

```
['Brain Tumor.csv', 'bt_dataset_t3.csv', 'Brain Tumor']
```

```
dataset_dir = os.path.join(path, 'Brain Tumor')
print(dataset_dir)
print(os.listdir(dataset_dir))

files_list = list(paths.list_images(dataset_dir))
print(f"Total Images: {len(files_list)}")

def natural_sort_key(file_name):
    return [int(text) if text.isdigit() else text.lower() for text in re.split(r'(\

files_list = sorted(files_list, key=natural_sort_key)

print(files_list[0])
```

```
/Users/grantreidy/.cache/kagglehub/datasets/jakeshbohaju/brain-tumor/versions/3/Bra
['Brain Tumor']
Total Images: 3762
/Users/grantreidy/.cache/kagglehub/datasets/jakeshbohaju/brain-tumor/versions/3/Bra
```

```
csv_dir = os.path.join(path, 'Brain Tumor.csv')
df = pd.read_csv(csv_dir)

label_list = df['Class'].tolist()
print(label_list[0])
```

```
0
```

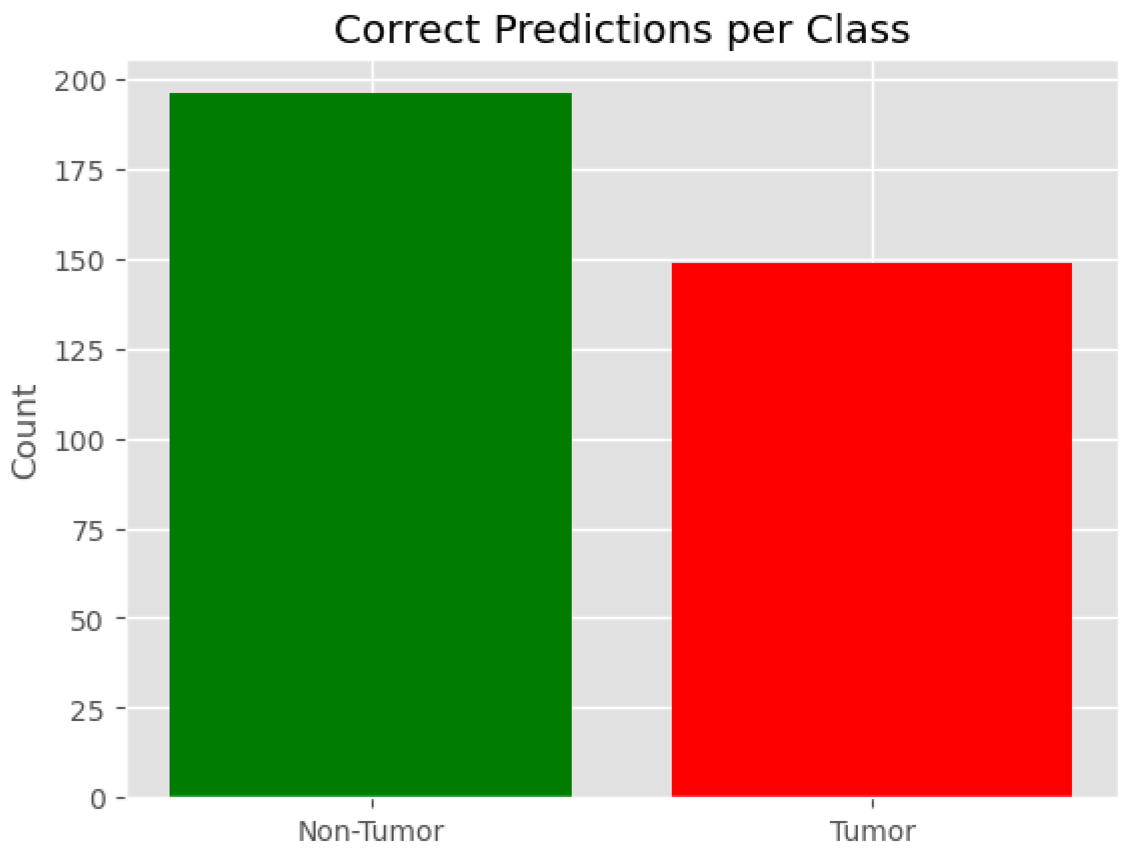
```
# files_list = files_list[:100]
# label_list = label_list[:100]
# len(files_list)
```

```
image_list = []

for img_file in files_list:
    img_array = cv2.imread(img_file)
    # img_resized = cv2.resize(img_array, (224, 224))
    img_resized = cv2.resize(img_array, (64, 64))
    img_flatten = img_resized.flatten()
    image_list.append(img_flatten)

image_data = np.array(image_list) / 255.0
label_data = np.array(label_list)

print(label_data)

a = np.array(label_data)

print(np.sum(a))
```

```
[0 0 1 ... 0 0 0]
1683
```

```
X_train, X_test, y_train, y_test = train_test_split(
    image_data, label_data, test_size=0.10, random_state=42, stratify=label_data
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_model = SVC(kernel="rbf", C=1.0, probability=True, random_state=42)

svm_model.fit(X_train, y_train)

predictions = svm_model.predict(X_test)

print(y_test)
print(predictions)

print(classification_report(y_test, predictions))


conf_matrix = confusion_matrix(y_test, predictions)
print(conf_matrix)

total_entries = sum(sum(conf_matrix))
model_accuracy = (conf_matrix[0, 0] + conf_matrix[1, 1]) / total_entries
print(f"Final Model Accuracy: {model_accuracy:.4f}")

plt.style.use("ggplot")
plt.figure()
plt.bar(["Non-Tumor", "Tumor"], conf_matrix.diagonal(), color=['green', 'red'])
plt.title("Correct Predictions per Class")
plt.ylabel("Count")
plt.show()
```

```
[0 0 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 1 1 0 0 1 0 0
 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0
 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 1 1
 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 1
 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 0 1
 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1
 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 1 0 0
 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 1 1 1 0 0 1 0
 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1
 1 1 1 0 0 0 0]
[0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 1 1 1 0 1 1 0 0 1 0 0
 1 0 1 1 0 1 1 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0
 0 1 0 0 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1
 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 1
 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 1 0 0 1
 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 1
 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 0
 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 1 1 1 0 1 0 0
 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0
 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 0 1
 1 1 1 0 0 0 0]
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       208
           1       0.93      0.88      0.90       169

    accuracy                           0.92       377
   macro avg       0.92      0.91      0.91       377
weighted avg       0.92      0.92      0.91       377

[[196  12]
 [ 20 149]]
Final Model Accuracy: 0.9151
```



Summary of SVM Results:

Classification Report:

| | Precision | Recall | F1-Score | Support | |————–|———–|——–|———-|———| | No (0) | 0.91 | 0.94 | 0.92 | 208 | | Yes (1) | 0.93 | 0.88 | 0.90 | 169 | | Accuracy | | | 0.92 | 377 | | Macro Avg | 0.92 | 0.91 | 0.91 | 377 | | Weighted Avg | 0.92 | 0.92 | 0.91 | 377 |

Analysis

The model achieved strong performance with an accuracy of 92%, demonstrating high precision, recall, and F1-scores for both classes. For the “No” (0) class, the model recorded 91% precision, 94% recall, and an F1-score of 92%, while the “Yes” (1) class achieved 93% precision, 88% recall, and an F1-score of 90%. The lower recall for the “Yes” class indicates that some true positives were misclassified as “No,” which could be significant depending on the application’s context. The macro-average F1-score of 91% highlights balanced class performance, and the weighted-average F1-score of 91% confirms the model accounts well for class proportions. However, addressing slightly lower recall for the “Yes” class might involve strategies like balancing the dataset or adjusting thresholds. Finally, testing the model on a larger, independent dataset could further validate its robustness and ensure generalizability to real-world scenarios.

Model Performance Comparison

Metric	Random Forest	CNN	SVM
Accuracy	96%	96%	92%
Precision (No)	100%	91%	91%

Metric	Random Forest	CNN	SVM
Recall (No)	90%	100%	94%
F1-Score (No)	95%	95%	92%
Precision (Yes)	94%	100%	93%
Recall (Yes)	100%	94%	88%
F1-Score (Yes)	97%	97%	90%
Macro Average	96% Precision,	95% Precision,	92% Precision,
	95% Recall,	97% Recall,	91% Recall,
	96% F1-Score	96% F1-Score	91% F1-Score

Observations

1. Random Forest:

1. Achieved the highest precision (100%) for classifying “No” cases.
2. Delivered the best overall recall for “Yes” cases (100%).
3. Accuracy of 96%, indicating strong performance.

2. CNN:

1. Performed equally well as Random Forest in terms of accuracy (96%).
2. Highest precision for “Yes” cases (100%), but recall was slightly lower (94%).

3. SVM:

1. Slightly lower accuracy (92%) compared to CNN and Random Forest.
2. Balanced performance with good precision (92%), recall (91%), and F1-score (91%) across both classes.
3. Performance on “Yes” cases (88% recall) needs improvement.

Recommendations

- The Random Forest and CNN models clearly perform the best in terms of accuracy and F1-score. Random Forest does a partticualrly good job at classifying “No” cases, while CNN is better at in precision for “Yes” cases.
- SVM performance is slighlty worse than other models. It may be useful to tune the hyperparmaters more such as the regularization paramter, gamma, degree, etc. But, as seen in the visualization, did a better job at classifying non tumor images compared to tumor images.

Next Steps

1. Dataset Expansion:
1. Increase the training dataset size using data augmentation or acquire new data to enhance the model’s robustness and reduce overfitting, especially critical due to small support sizes for Random Forest and CNN.

2. Ensemble Approaches:
1. Combine predictions from the three models to leverage their strengths. For instance, Random Forest can focus on “No” cases while CNN targets “Yes” cases.

3. Testing on Larger Datasets:

1. Validate the models on a significantly larger and diverse dataset to test generalizability and ensure performance consistency.

4. Feature Engineering:
   1. Explore additional features or apply dimensionality reduction techniques to improve SVM's performance.

Summary of Contributions

Member	Contributions
Keshav Garg	Preprocessing Steps, SVM Model and Metrics, Final Slides
Grant Reidy	CNN Model, Website, Markdown, CNN Performance Metrics
Vraj Kothari	Random Forest Model, Final Slides, Final Video
Aditya Tagore	Model Generation and Algorithm completion, Final Report
Anshul Makwan	Random Forest Generation, Final Slides, and Report

Gantt Chart

Gantt Chart

References

1. Sandeep Kumar Mathivanan, Sridevi Sonaimuthu, Sankar Murugesan, Hariharan Rajadurai, Basu Dev Shivahare, Mohd Asif Shah, "Employing deep learning and transfer learning for accurate brain tumor detection", Scientific Reports, March 2024, Mathivanan2024.

2. IBM, "Convolutional Neural Networks", accessed October 4, 2024, IBM CNN.

3. Deepak Sharma, P. M. Ameer, "Brain tumor classification using deep CNN features via transfer learning", Computers in Biology and Medicine, 111:103345, 2019.

4. M. Havaei, A. Davy, D. Warde-Farley, J. Biard, A. Courville, Y. Bengio, C. Pal, P. M. Jodoin, H. Larochelle, "Brain tumor segmentation with Deep Neural Networks", Medical Image Analysis, 35:18-31, 2017.

5. Jakesh Bohaju, "Brain Tumor Dataset", Kaggle Dataset.

6. scikit-learn developers, "Model Evaluation: quantifying the quality of predictions", 2023, accessed October 4, 2024, Scikit-learn Evaluation.

7. Fiona M. Walter et al., "Missed opportunities for diagnosing brain tumours in primary care: a qualitative study of patient experiences", British Journal of General Practice, 69(681):e224-e235, 2019.