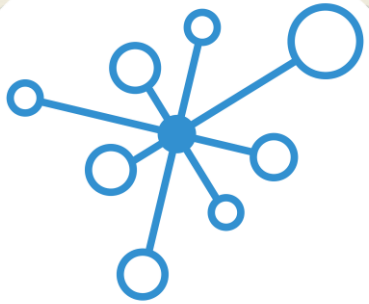


BRAIN TUMOR DETECTION USING CNN

Presented By –
Kunal More
Student at Develearn

INTRODUCTION

INTRODUCTION

We provide the brain tumor detection project code and dataset for you. This dataset is already divided into training and testing. The train folder contains the images which will be passed as a training dataset. The test folder contains the images which will be passed as a test dataset. Then we finally predict the result by passing an image from our testing dataset.

Brain Tumor

The source of brain tumors can be traced back to aberrant cells forming in the brain, some of which are precancerous, some of which are cancerous or damaging, and others which are innocuous or noncancerous.

Tumors are classified into three types: those that begin in the brain and those that are subsequent cancerous growths that can spread to different areas of the human body; in such circumstances, the carcinoma is said to have metastasized in the patient and is most lethal, with a very poor survival rate.

Classification of Brain Tumors

Brain tumors can be classified into three main types, firstly there are pituitary tumors secondly meningioma tumor and the third one is glioma tumor that are cancerous in nature.

- The characteristics of tumors are as follows:
  a) Majority of benign tumors are identified and diagnosed using CT and MRI brain scans
  b) Tumors usually have a very slow rate of growth and do not spread into neighboring tissues.
  c) It grows slowly and does not invade surrounding tissues or spread to other organs.
  d) They can be easily seen of CT scan by identifying their patently visible edges and boundaries

Classification of Brain Tumors

1) *Malignant Tumor:* Malignant brain tumors are cancerous cells and often having boundaries and edges that are not easily visible or identified. With highly rapid growth rate, Malignant tumors are the most life-threatening growths due their tendency to aggressively spread and invade the surrounding tissues.

- The characteristics of a malignant tumors are as follows:
  a) Rapid development and inclination to spread to various regions of the brain and spinal cord, making them extremely deadly with a high mortality and poor survival rate.
  b) Tumors can be graded on various levels, for instance grade 1 and 2 tumors can be either harmless or cancerous, but grade 3 and 4 are considered to be definite malignant growths.

PROBLEM STATEMENT

Motivation

Diagnosing tumors in medical imaging is time consuming due to its manual nature as it mainly relies on human ability and judgment. Specialists in this field, such as radiologists, examine images from CT scans, MRIs, and PET scans and make decisions on which treatment depends. This assiduous process takes a couple of hours to complete. Automation of the detection process helps to cut down a significant amount of time and effort needed.

Objective

The primary goal of this work is to create a model that can predict whether or not MRI images include cancer. We created and trained a model that could detect the tumor, presenting an efficient and effective way for assisting in the segmentation and identification of brain tumors that eliminates the need for manual labor. Finally, when we compared the outcomes of all tests, we discovered that certain models performed better in terms of accuracy and loss metrics.

METHODOLOGY

Dataset

The Database was recieved from **Develearn**, named ‘Brain MRI Images for brain tumor Detection The dataset is in two folder testing and training already with total size 3264 images Brain MRI Images in the folders training and testing

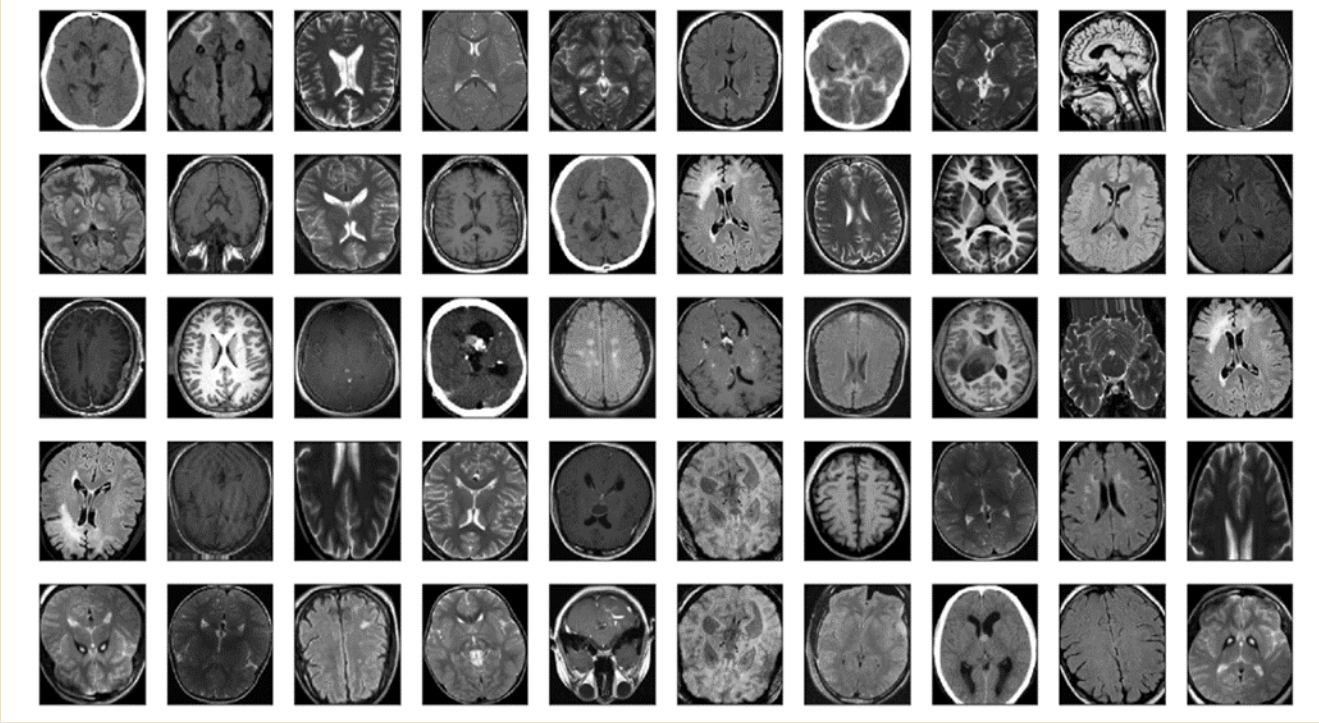


Image Preprocessing

Pre-processing is important in order to create a seamless training experience because the MRI scans differ in intensity, contrast, and size. In the first preprocessing step, warping as well as cropping is performed, which will prepare the input picture.

During warping, the supplied image is compared to the primary subject in the window. The image's outermost boundary is set in order for the subject to remain intact after cropping. Because the photographs in the dataset are of varying sizes, the image is reshaped to 128 x 128. then we convert into numpy array

Importing Libraries

```
import os
import numpy as np
from PIL import Image
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split
```

Load and Preprocess Data

```
train_dir = r"E:\Brain Tumor Classification\Training"
test_dir = r"E:\Brain Tumor Classification\Testing"
image_size = (128, 128)
```

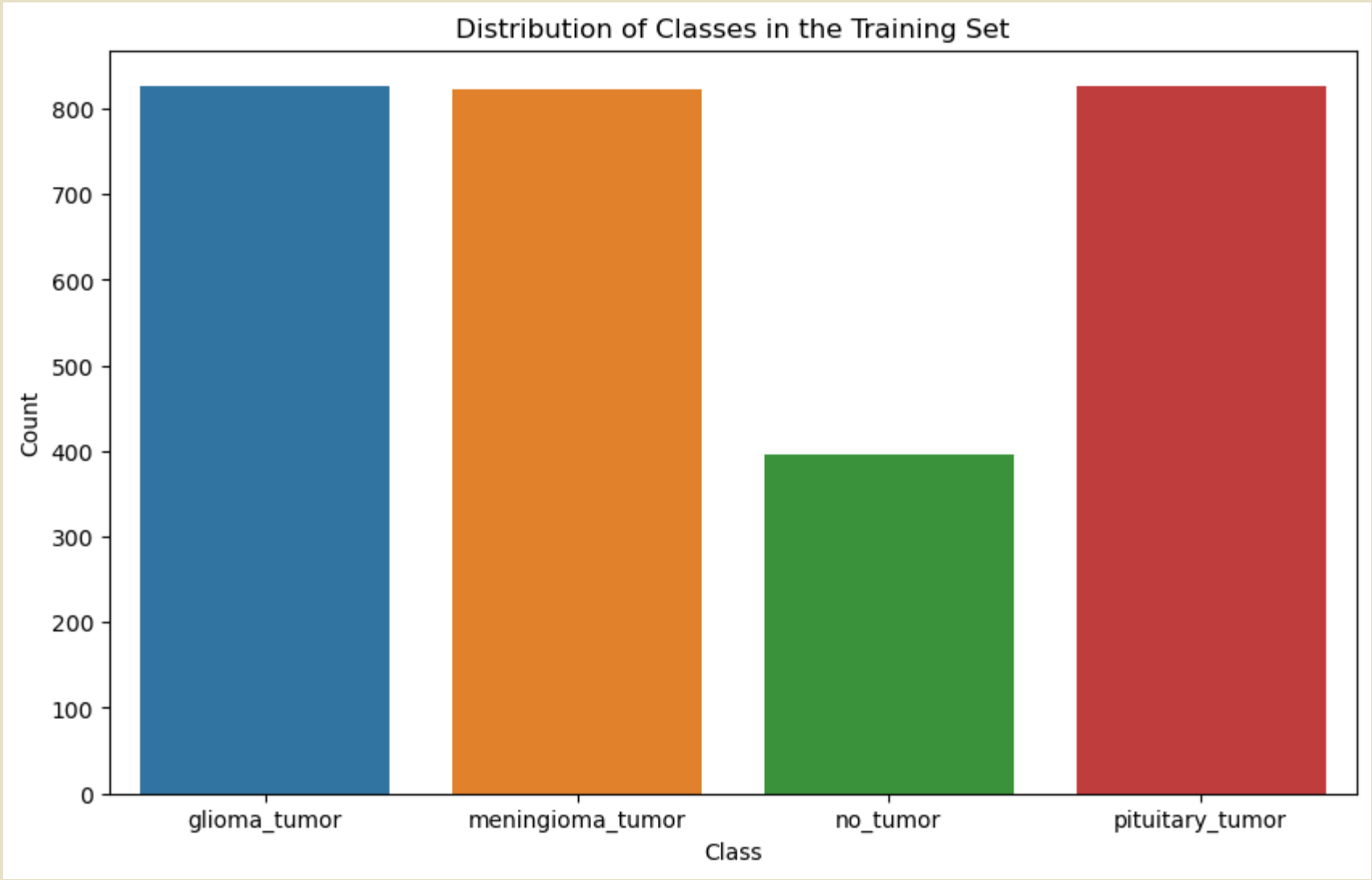
```
def load_and_resize_images_from_folder(folder, image_size):
    images = []
    labels = []
    for class_name in os.listdir(folder):
        class_dir = os.path.join(folder, class_name)
        for filename in os.listdir(class_dir):
            img_path = os.path.join(class_dir, filename)
            img = Image.open(img_path).resize(image_size)
            if img is not None:
                images.append(np.array(img))
                labels.append(class_name)
    return np.array(images), np.array(labels)
```

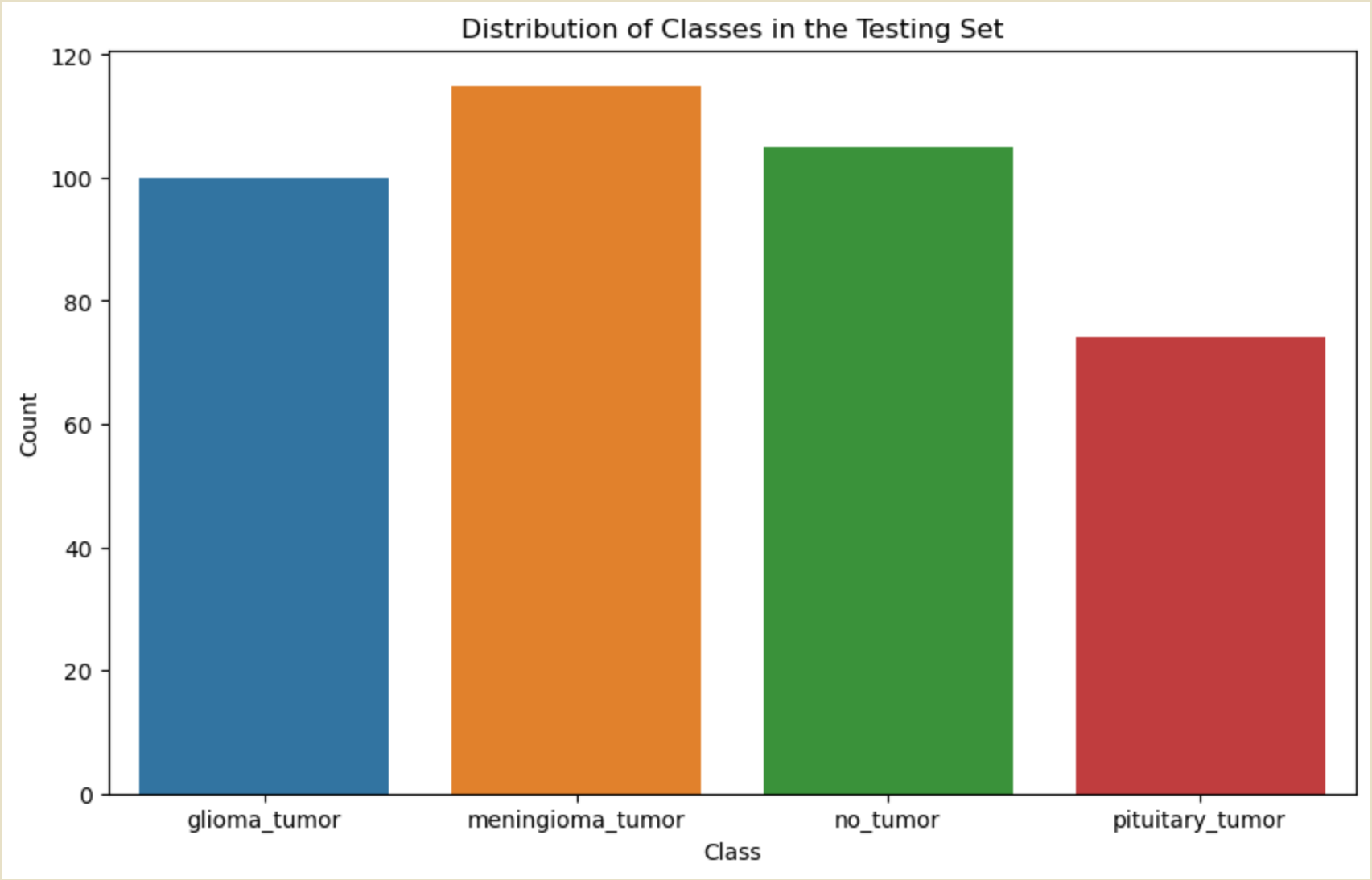
```
train_images, train_labels = load_and_resize_images_from_folder(train_dir, image_size)
test_images, test_labels = load_and_resize_images_from_folder(test_dir, image_size)

train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

assert len(train_images) == len(train_labels), "Mismatch between images and labels in training data"
assert len(test_images) == len(test_labels), "Mismatch between images and labels in testing data"
```

Data Visualization


Distribution of Classes in the Training Set
Count
0
100
200
300
400
500
600
700
800
glioma_tumor
meningioma_tumor
no_tumor
pituitary_tumor
Class

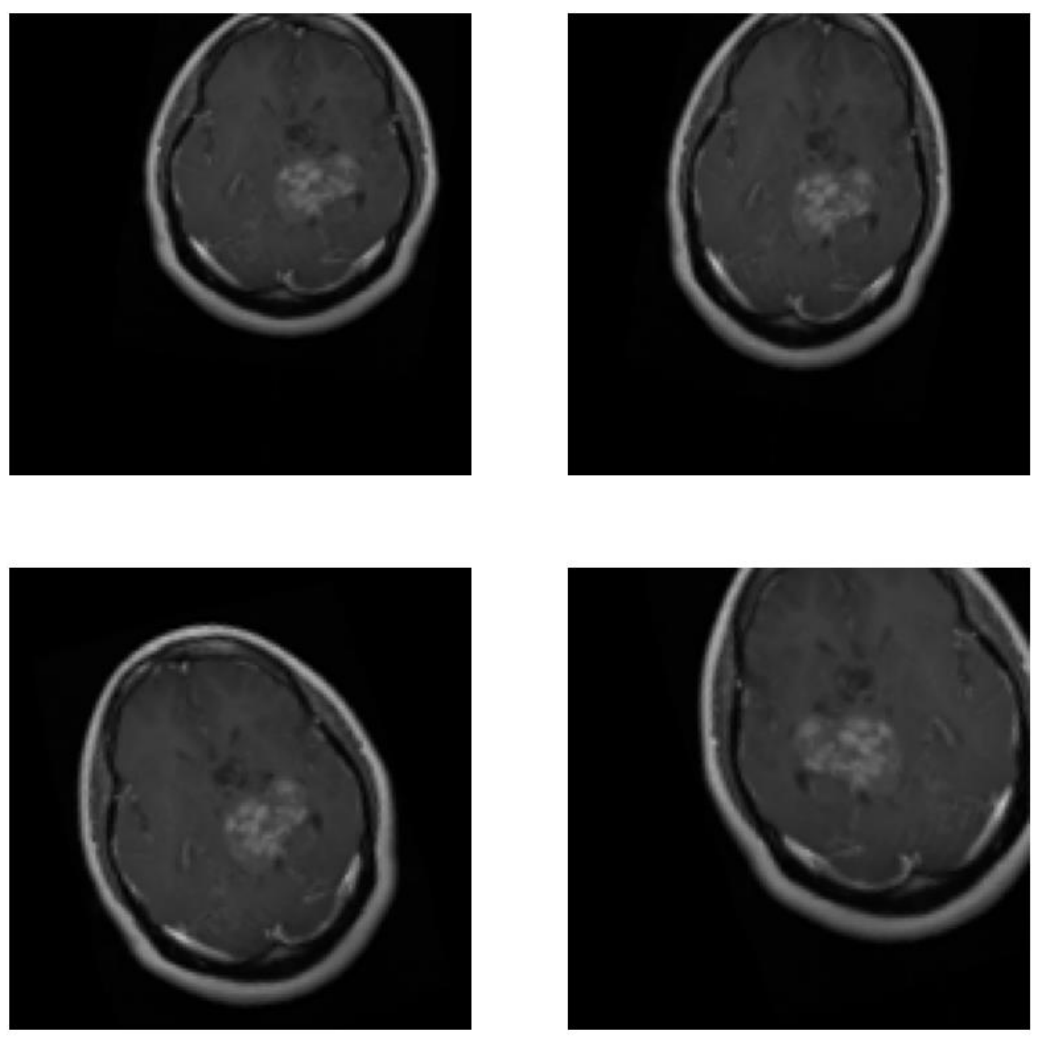

Distribution of Classes in the Testing Set
120
100
80
60
40
20
0
Count
glioma_tumor
meningioma_tumor
no_tumor
pituitary_tumor
Class

Data Augmentation

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Check the output of the data generator
sample_image = train_images[0] * 255
sample_image = sample_image.astype('uint8')
sample_image = sample_image.reshape((1,) + sample_image.shape)

i = 0
plt.figure(figsize=(10, 10))
for batch in datagen.flow(sample_image, batch_size=1):
    plt.subplot(2, 2, i + 1)
    img = batch[0].astype('uint8')
    plt.imshow(img)
    plt.axis('off')
    i += 1
    if i % 4 == 0:
        break
plt.show()
```



VGG16

VGG16 is a convolutional neural network architecture known for its simplicity and effectiveness in image classification. It consists of 13 convolutional layers and 3 fully connected layers, totaling 16 layers. The network uses 3x3 convolutional filters and max pooling layers to extract and reduce spatial dimensions of features.

While **VGG16** performs well in image recognition tasks, its deep architecture makes it computationally expensive compared to newer models. Despite this, **VGG16** remains a widely used benchmark in deep learning for image classification.

Build and compile the model using transfer learning with VGG16

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_size[0], image_size[1], 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(train_labels)), activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	?	14,714,688
flatten (Flatten)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
batch_normalization (BatchNormalization)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)
batch_normalization_1 (BatchNormalization)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0
dense_2 (Dense)	?	0 (unbuilt)

Total params: 14,714,688 (56.13 MB)

Trainable params: 0 (0.00 B)

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

Split data for training and validation

```
X_train, X_val, y_train, y_val = train_test_split(train_images, train_labels, test_size=0.2, random_state=42)
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)

class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_encoded),
    y=y_train_encoded
)
class_weights = {i: class_weights[i] for i in range(len(class_weights))}

# Train the model
results = model.fit(
    datagen.flow(X_train, y_train_encoded, batch_size=32),
    epochs=25,
    validation_data=(X_val, y_val_encoded),
    class_weight=class_weights
)
```

Train the model

```python
results = model.fit(
    datagen.flow(X_train, y_train_encoded, batch_size=32),
    epochs=50,  # Increased number of epochs
    validation_data=(X_val, y_val_encoded),
    class_weight=class_weights
)
```

```
Epoch 1/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.4639 - loss: 1.6459 - val_accuracy: 0.6063 - val_loss: 1.2543
Epoch 2/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 202s 3s/step - accuracy: 0.6527 - loss: 0.9944 - val_accuracy: 0.6707 - val_loss: 1.0441
Epoch 3/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 204s 3s/step - accuracy: 0.6902 - loss: 0.8871 - val_accuracy: 0.7352 - val_loss: 0.7979
Epoch 4/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 194s 3s/step - accuracy: 0.6903 - loss: 0.7688 - val_accuracy: 0.7892 - val_loss: 0.6233
Epoch 5/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 188s 3s/step - accuracy: 0.7232 - loss: 0.7314 - val_accuracy: 0.7927 - val_loss: 0.5543
Epoch 6/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 191s 3s/step - accuracy: 0.7224 - loss: 0.6966 - val_accuracy: 0.7683 - val_loss: 0.6058
Epoch 7/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 189s 3s/step - accuracy: 0.7566 - loss: 0.6057 - val_accuracy: 0.7840 - val_loss: 0.6211
Epoch 8/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 189s 3s/step - accuracy: 0.7559 - loss: 0.5843 - val_accuracy: 0.7822 - val_loss: 0.5707
Epoch 9/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 190s 3s/step - accuracy: 0.7526 - loss: 0.6000 - val_accuracy: 0.6916 - val_loss: 0.9421
Epoch 10/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 188s 3s/step - accuracy: 0.7781 - loss: 0.5566 - val_accuracy: 0.7631 - val_loss: 0.6798
Epoch 11/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 189s 3s/step - accuracy: 0.7931 - loss: 0.5208 - val_accuracy: 0.7997 - val_loss: 0.5740
```

Evaluate the model

```
val_loss, val_accuracy = model.evaluate(X_val, y_val_encoded)
print(f"Validation Accuracy: {val_accuracy}")
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.8180 - loss: 0.5464
Validation Accuracy: 0.8257839679718018
```

```
test_labels_encoded = label_encoder.transform(test_labels)
test_loss, test_accuracy = model.evaluate(test_images, test_labels_encoded)
print(f"Test Accuracy: {test_accuracy}")
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 22s 2s/step - accuracy: 0.4096 - loss: 3.0912
Test Accuracy: 0.6218274235725403
```

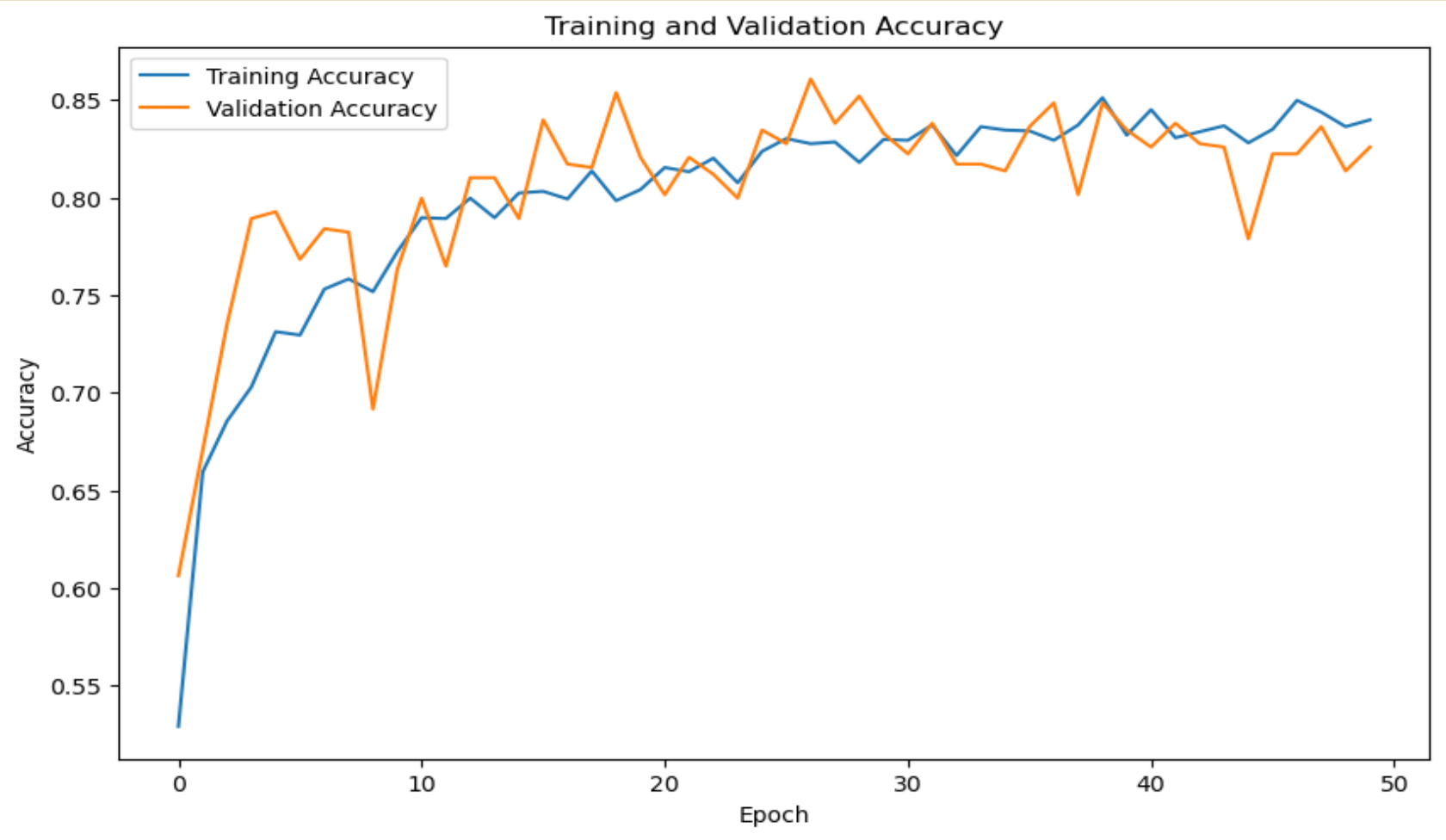
```
y_pred = model.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)

conf_matrix = confusion_matrix(test_labels_encoded, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print(classification_report(test_labels_encoded, y_pred_classes, target_names=label_encoder.classes_))
```

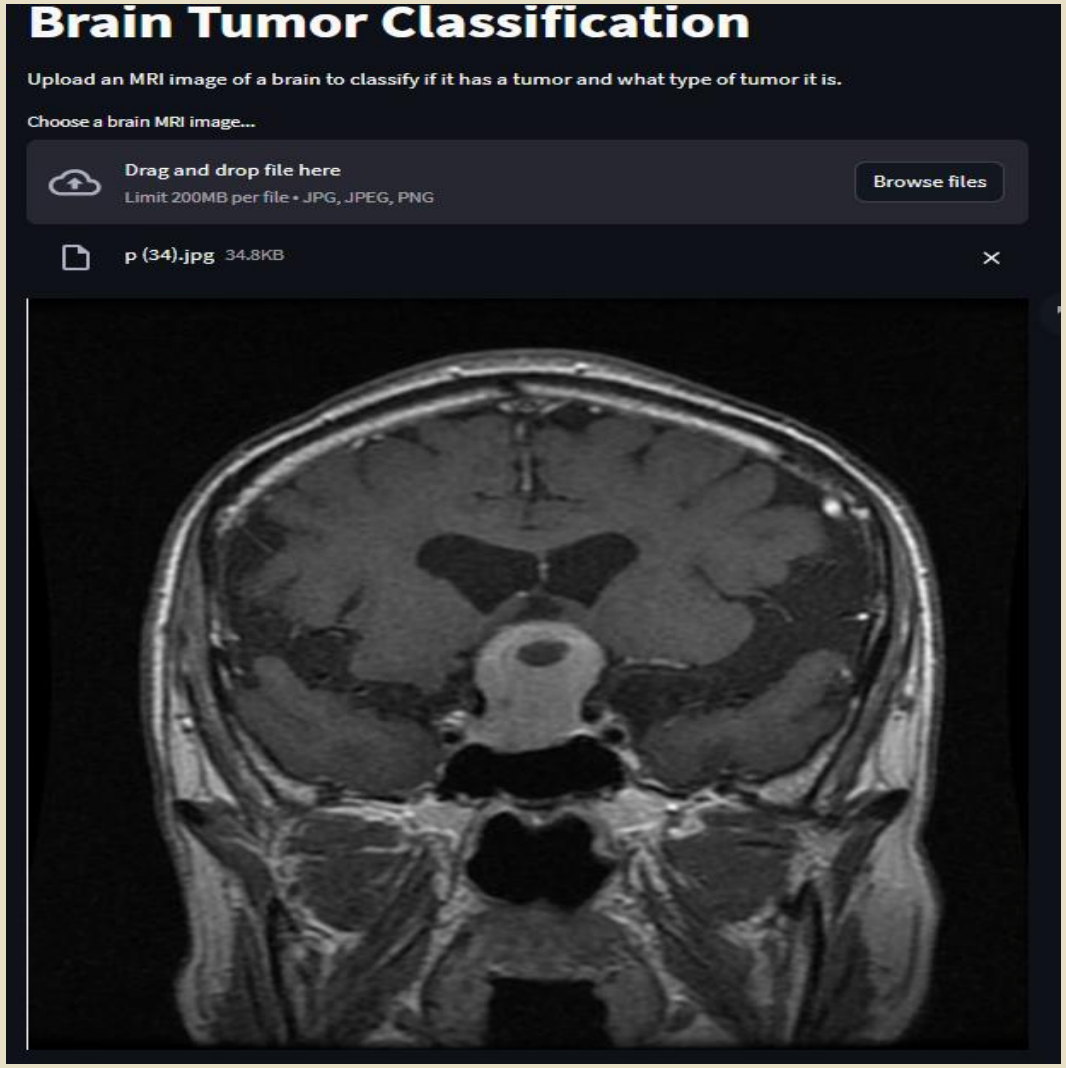
```
13/13 ━━━━━━━━━━━━━━━━━━━━ 30s 2s/step
```

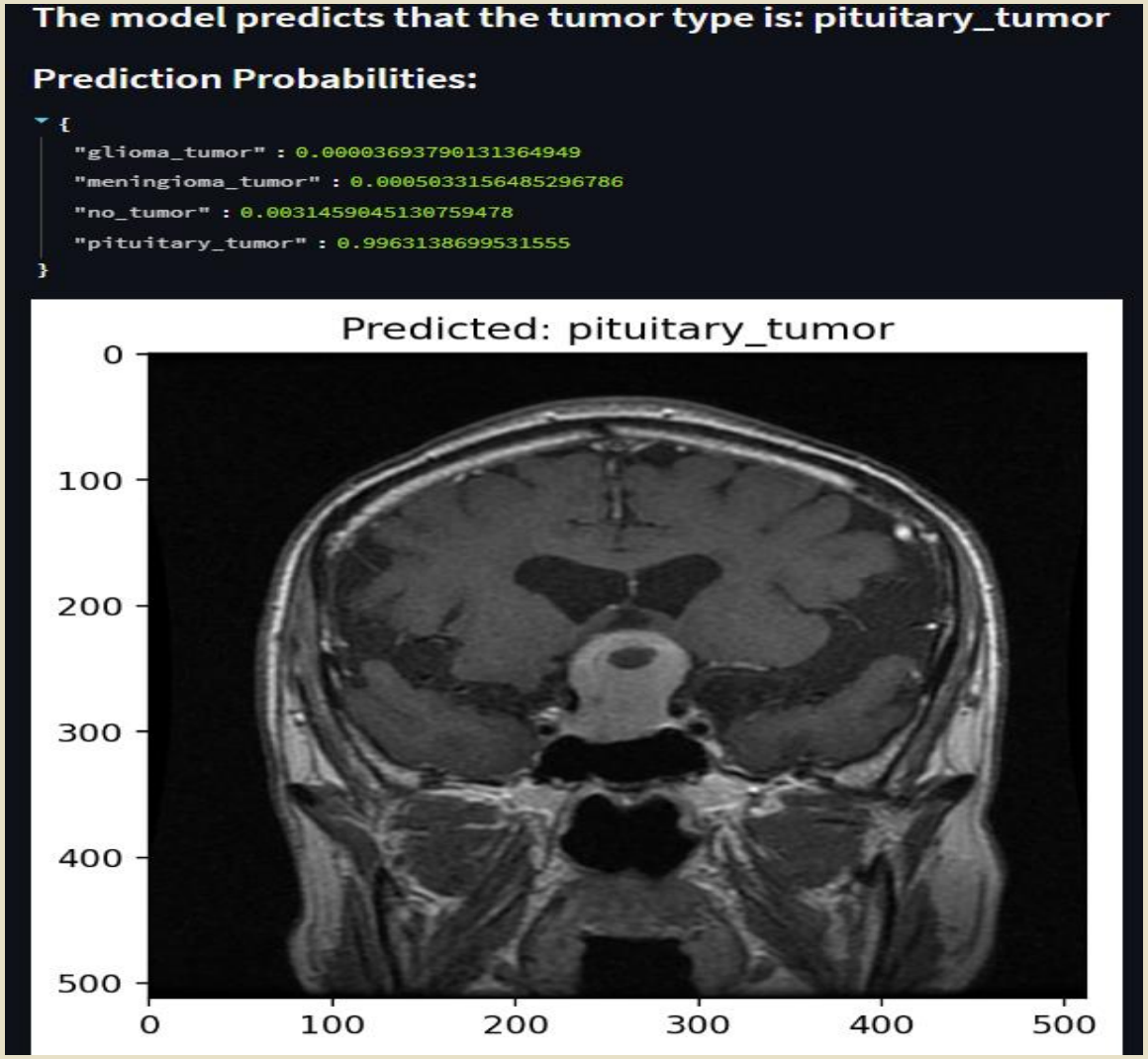
Training and validation accuracy



Training and validation loss

Prediction With Deployment





CONCLUSION

CONCLUSION

CNNs are excellent tools for detecting brain cancers in MRI images. In the 50th epoch, this investigation validation accuracy of 82%. The model performance can be improved in the future we have access to additional amount of data and more powerful hardware to handle computations involving such a large amount of data.