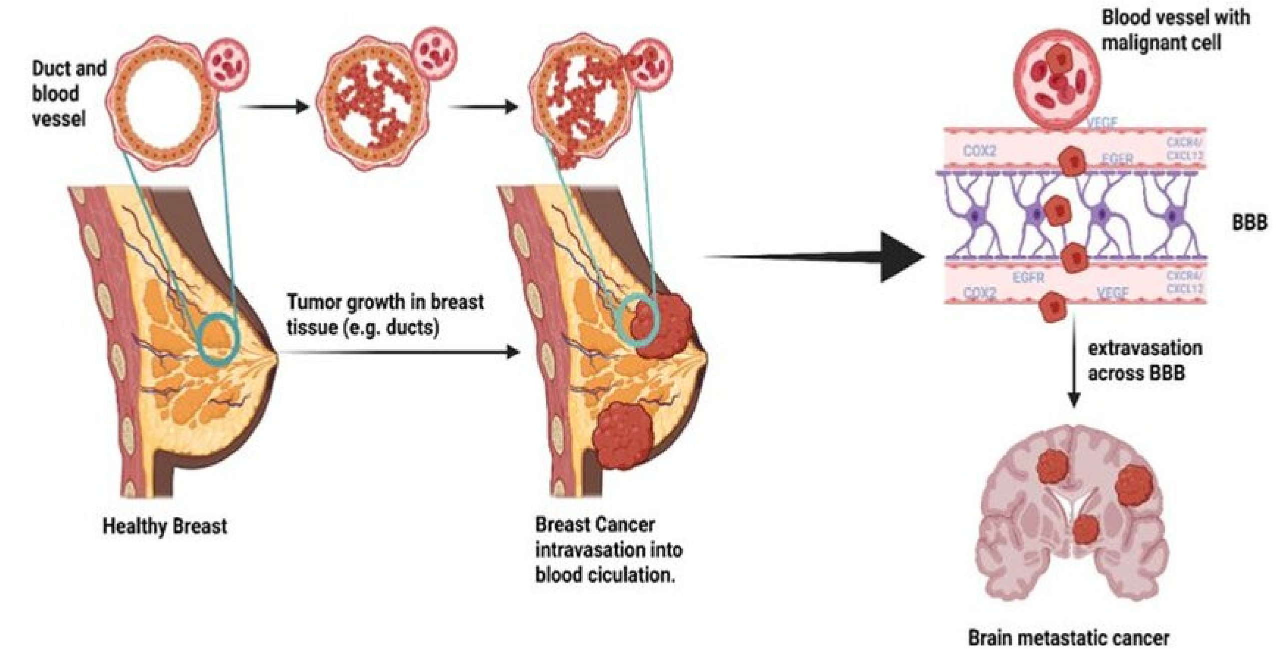
# Brain Breast Cancer Classification




```
import pandas as pd
import os

base_paths = [
    ("/kaggle/input/brain-breast-tumor/Dataset/Brain scans/No tumor/Train", "No tumor", "Train"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Brain scans/No tumor/Test", "No tumor", "Test"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Brain scans/Tumor/TRAIN", "Tumor", "Train"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Brain scans/Tumor/TEST", "Tumor", "Test"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Breast scans/benign/Train", "benign", "Train"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Breast scans/benign/Test", "benign", "Test"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Breast scans/malignant/Train", "malignant", "Train"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Breast scans/malignant/Test", "malignant", "Test"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Breast scans/normal/Train", "normal", "Train"),
    ("/kaggle/input/brain-breast-tumor/Dataset/Breast scans/normal/Test", "normal", "Test")
]
```

```python
def collect_image_data():
    image_paths = []
    labels = []
    splits = []
    for path, label, split in base_paths:
        if not os.path.exists(path):
            print(f"Warning: Directory {path} does not exist.")
            continue
        for img_name in os.listdir(path):
            img_path = os.path.join(path, img_name)
            if os.path.isfile(img_path) and 
img_name.lower().endswith(('.jpg', '.jpeg', '.png')):
                image_paths.append(img_path)
                labels.append(label)
                splits.append(split)
    return image_paths, labels, splits

image_paths, labels, splits = collect_image_data()

df = pd.DataFrame({
    'Image_Path': image_paths,
    'Label': labels,
    'Split': splits
})

df.head()
```

| | Image_Path | Label | Split |
|---|---|---|---|
| 0 | /kaggle/input/brain-breast-tumor/Dataset/Brain... | No tumor | Train |
| 1 | /kaggle/input/brain-breast-tumor/Dataset/Brain... | No tumor | Train |
| 2 | /kaggle/input/brain-breast-tumor/Dataset/Brain... | No tumor | Train |
| 3 | /kaggle/input/brain-breast-tumor/Dataset/Brain... | No tumor | Train |
| 4 | /kaggle/input/brain-breast-tumor/Dataset/Brain... | No tumor | Train |

```python
df.tail()
```

| | Image_Path | Label | Split |
|---|---|---|---|
| 2640 | /kaggle/input/brain-breast-tumor/Dataset/Breas... | normal | Test |
| 2641 | /kaggle/input/brain-breast-tumor/Dataset/Breas... | normal | Test |
| 2642 | /kaggle/input/brain-breast-tumor/Dataset/Breas... | normal | Test |
| 2643 | /kaggle/input/brain-breast-tumor/Dataset/Breas... | normal | Test |
| 2644 | /kaggle/input/brain-breast-tumor/Dataset/Breas... | normal | Test |

```python
df.shape
```

```
(2645, 3)
```

```python
df.columns
```

```
Index(['Image_Path', 'Label', 'Split'], dtype='object')
```

```
df.duplicated().sum()
```

```
0
```

```
df.isnull().sum()
```

```
Image_Path    0
Label         0
Split         0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2645 entries, 0 to 2644
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Image_Path  2645 non-null   object
 1   Label       2645 non-null   object
 2   Split       2645 non-null   object
dtypes: object(3)
memory usage: 62.1+ KB
```

```
df['Label'].unique()
```

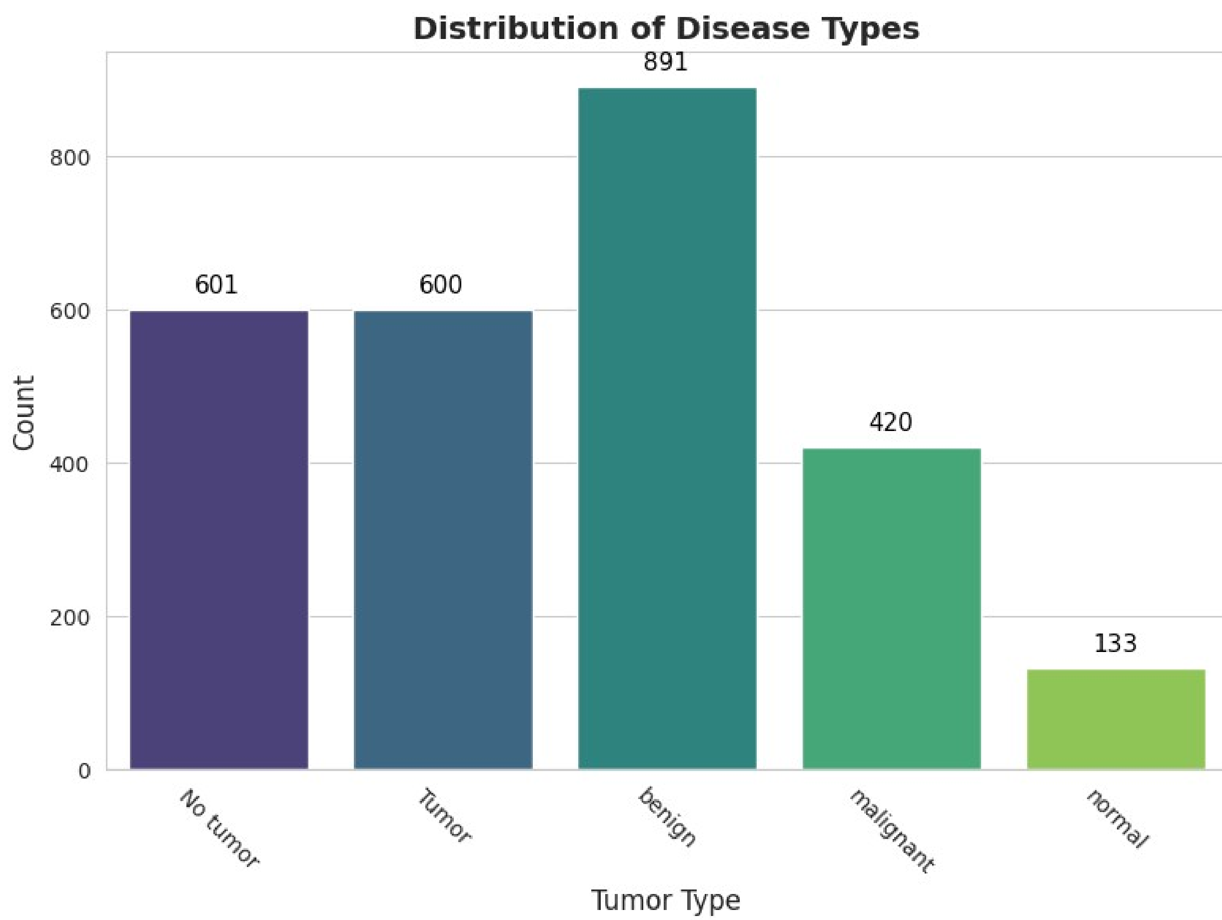
```
array(['No tumor', 'Tumor', 'benign', 'malignant', 'normal'],
dtype=object)
```

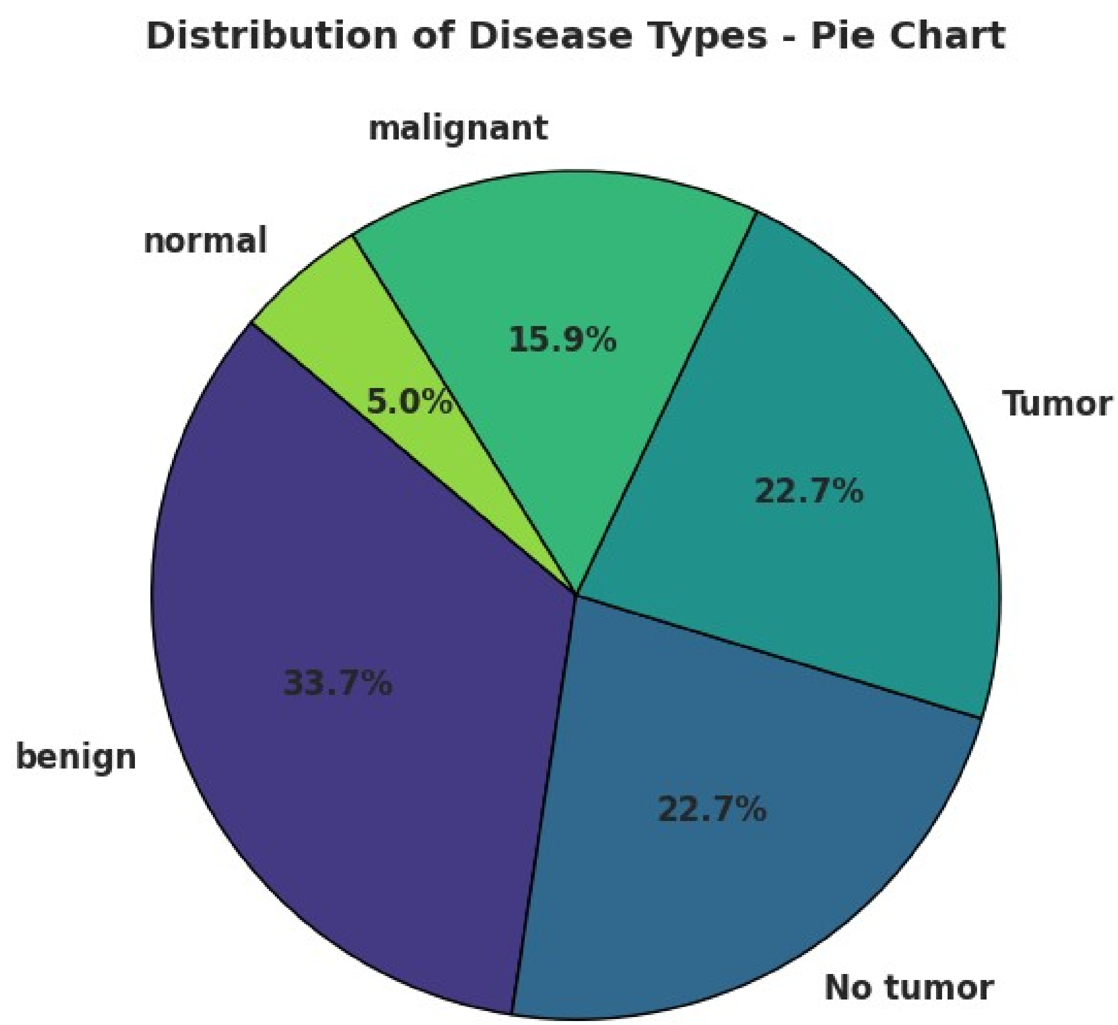
```
df['Label'].value_counts()
```

```
Label
benign       891
No tumor     601
Tumor        600
malignant    420
normal       133
Name: count, dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df, x="Label", palette="viridis", ax=ax)
ax.set_title("Distribution of Disease Types", fontsize=14,
fontweight='bold')
ax.set_xlabel("Tumor Type", fontsize=12)
ax.set_ylabel("Count", fontsize=12)
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
```

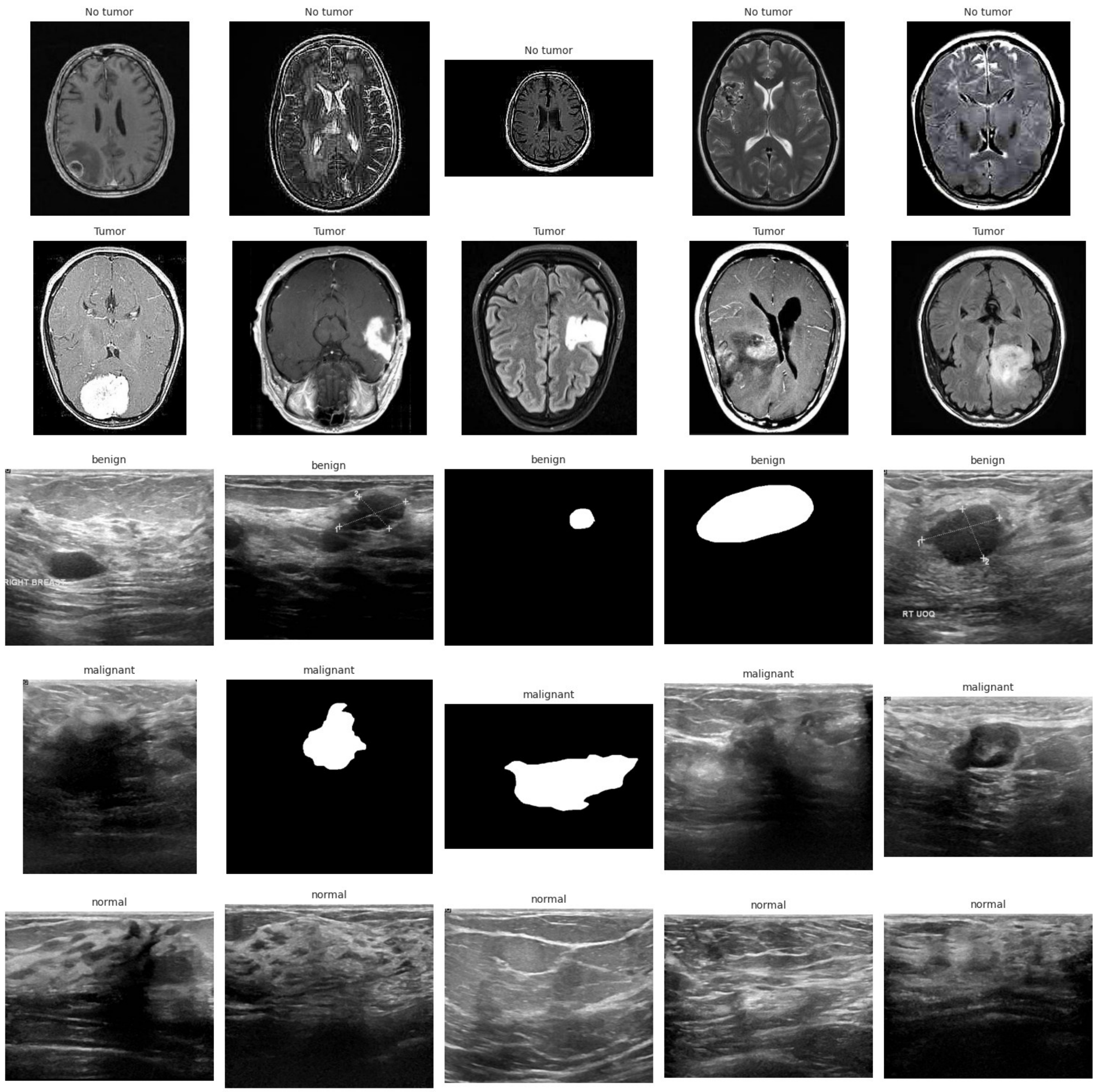
```
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=11, color='black',
                xytext=(0, 5), textcoords='offset points')
plt.xticks(rotation=-45)
plt.tight_layout()
plt.show()

label_counts = df["Label"].value_counts()
fig, ax = plt.subplots(figsize=(8, 6))
colors = sns.color_palette("viridis", len(label_counts))
ax.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
       startangle=140, colors=colors, textprops={'fontsize': 12,
'weight': 'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})
ax.set_title("Distribution of Disease Types - Pie Chart", fontsize=14,
fontweight='bold')
plt.tight_layout()
plt.show()
```






```
import cv2

categories = df['Label'].unique()

n_cols = 5
n_rows = len(categories)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 3, n_rows *
3))
if n_rows == 1:
    axes = [axes]
else:
    axes = axes.flatten()

plot_idx = 0
```

```
for category in categories:
    category_paths = df[df['Label'] == category]
['Image_Path'].head(5).tolist()

    for img_path in category_paths:
        if plot_idx < len(axes):
            try:
                img = cv2.imread(img_path)
                if img is None:
                    axes[plot_idx].text(0.5, 0.5, 'Image not found',
                                        ha='center', va='center',
transform=axes[plot_idx].transAxes)
                else:
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    axes[plot_idx].imshow(img)
                axes[plot_idx].set_title(category, fontsize=10)
                axes[plot_idx].axis('off')
            except:
                axes[plot_idx].text(0.5, 0.5, 'Error loading',
                                    ha='center', va='center',
transform=axes[plot_idx].transAxes)
            plot_idx += 1

    while len(category_paths) < 5 and plot_idx < len(axes):
        axes[plot_idx].text(0.5, 0.5, 'No image',
                            ha='center', va='center',
transform=axes[plot_idx].transAxes)
        axes[plot_idx].set_title(category, fontsize=10)
        axes[plot_idx].axis('off')
        plot_idx += 1

plt.tight_layout()
plt.show()
```




```
from sklearn.utils import resample

categories = df['Label'].unique()

max_count = df['Label'].value_counts().max()

df_balanced = pd.DataFrame()

for category in categories:
    df_category = df[df['Label'] == category]
    df_category_resampled = resample(df_category,
                                     replace=True,
                                     n_samples=max_count,
```

```
                                    random_state=42)
    df_balanced = pd.concat([df_balanced, df_category_resampled],
ignore_index=True)

df_balanced = df_balanced.sample(frac=1,
random_state=42).reset_index(drop=True)

print(df_balanced['Label'].value_counts())

df = df_balanced
```

```
Label
malignant    891
No tumor     891
Tumor        891
normal       891
benign       891
Name: count, dtype: int64
```

```
df = df[['Image_Path', 'Label']]

import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, Dense, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.utils import class_weight

image_paths = df['Image_Path'].values
labels = df['Label'].values

label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
labels_onehot = tf.keras.utils.to_categorical(labels_encoded,
num_classes=5)
class_names = label_encoder.classes_

def load_and_preprocess_image(path):
    try:
        img = load_img(path, target_size=(64, 64))
        img = img_to_array(img) / 255.0
        return img
    except:
        return np.zeros((64, 64, 3))
```

```
images = np.array([load_and_preprocess_image(path) for path in
image_paths])

X_train, X_val, y_train, y_val = train_test_split(images,
labels_onehot, test_size=0.2, random_state=42)

class_weights = class_weight.compute_class_weight('balanced',
classes=np.unique(labels_encoded), y=labels_encoded)
class_weights = dict(enumerate(class_weights))

def augment_image(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, lower=0.9, upper=1.1)
    angle = tf.random.uniform([], -0.1745, 0.1745)
    image = tf.image.rot90(image, k=tf.cast(tf.math.floor(angle /
(np.pi / 2)), tf.int32))
    return image, label

train_dataset = tf.data.Dataset.from_tensor_slices((X_train,
y_train)).map(augment_image,
num_parallel_calls=tf.data.AUTOTUNE).batch(32).shuffle(buffer_size=100
0).prefetch(tf.data.AUTOTUNE)
val_dataset = tf.data.Dataset.from_tensor_slices((X_val,
y_val)).batch(32).prefetch(tf.data.AUTOTUNE)

model = Sequential([
    Input(shape=(64, 64, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(5, activation='softmax')
])

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

early_stopping = EarlyStopping(monitor='val_accuracy', patience=5,
restore_best_weights=True)

history = model.fit(train_dataset, validation_data=val_dataset,
epochs=50, class_weight=class_weights, callbacks=[early_stopping])

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.tight_layout()
plt.savefig('training_history.png')
plt.show()

val_predictions = model.predict(X_val)
val_predictions = np.argmax(val_predictions, axis=1)
val_true = np.argmax(y_val, axis=1)
cm = confusion_matrix(val_true, val_predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.show()

print("\nClassification Report:")
print(classification_report(val_true, val_predictions,
target_names=class_names))
accuracy = accuracy_score(val_true, val_predictions)
print(f"Validation Accuracy: {accuracy:.4f}")
```

```
Epoch 1/50
```

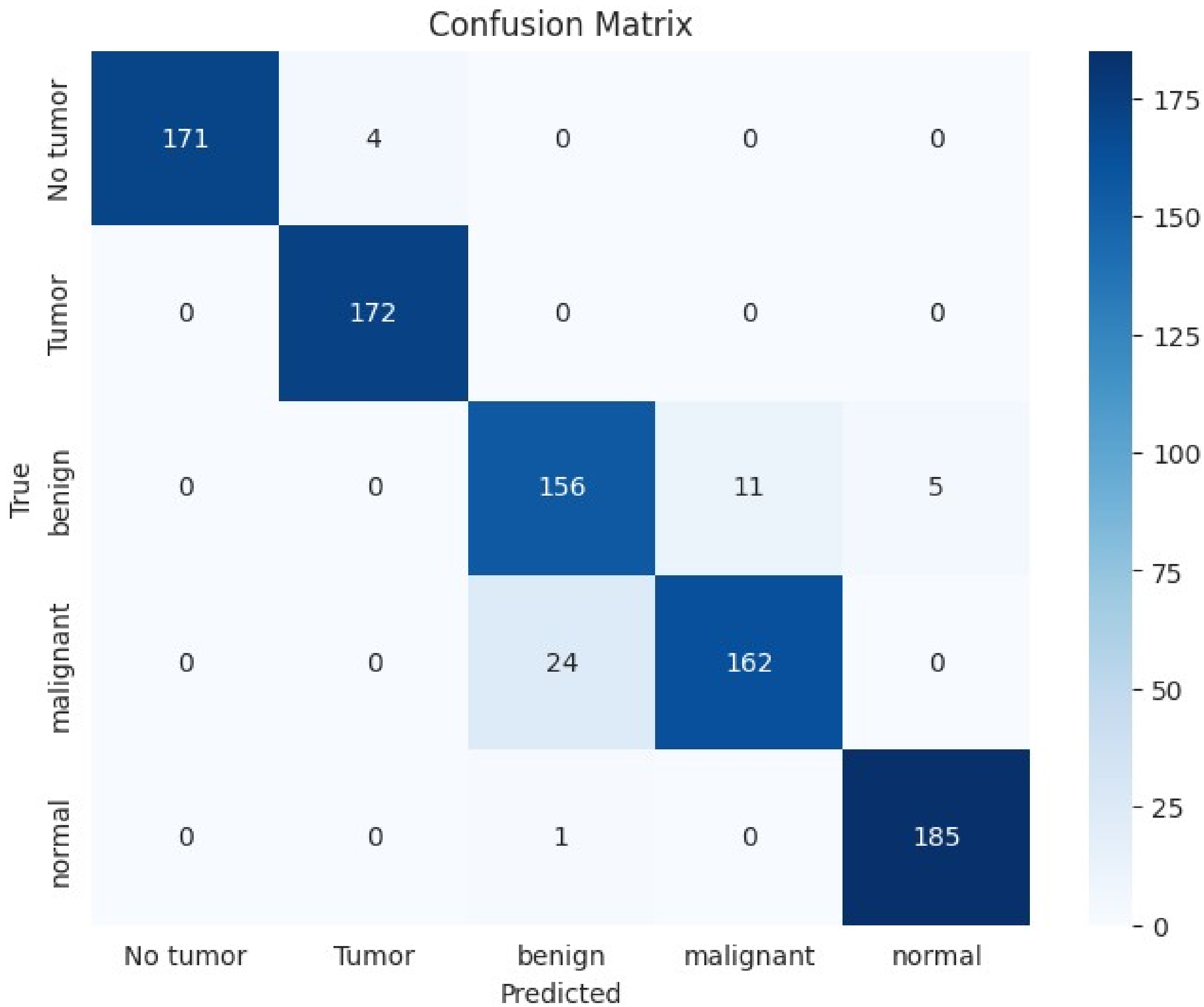
```
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1751450241.612510     126 service.cc:148] XLA service
0x7f2cd4002ba0 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1751450241.620094     126 service.cc:156]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1751450241.620117     126 service.cc:156]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
```

```
 30/112 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4731 - loss:
3.7331
```

```
I0000 00:00:1751450244.286056     126 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

112/112 ━━━━━━━━━━━━━━━━━━━━ 10s 32ms/step - accuracy: 0.5734 - loss:
2.2460 - val_accuracy: 0.2132 - val_loss: 7.8680
Epoch 2/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7727 - loss:
0.5649 - val_accuracy: 0.2076 - val_loss: 9.0292
Epoch 3/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8356 - loss:
0.4022 - val_accuracy: 0.4444 - val_loss: 4.8330
Epoch 4/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8948 - loss:
0.2738 - val_accuracy: 0.5735 - val_loss: 1.6305
Epoch 5/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8941 - loss:
0.2671 - val_accuracy: 0.7430 - val_loss: 0.8041
Epoch 6/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9126 - loss:
0.2309 - val_accuracy: 0.8676 - val_loss: 0.3325
Epoch 7/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9252 - loss:
0.2128 - val_accuracy: 0.8743 - val_loss: 0.3314
Epoch 8/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9293 - loss:
0.1772 - val_accuracy: 0.8900 - val_loss: 0.3118
Epoch 9/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9658 - loss:
0.1141 - val_accuracy: 0.9024 - val_loss: 0.2835
Epoch 10/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9582 - loss:
0.1174 - val_accuracy: 0.8979 - val_loss: 0.2526
Epoch 11/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9735 - loss:
0.0866 - val_accuracy: 0.9057 - val_loss: 0.3257
Epoch 12/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9743 - loss:
0.0813 - val_accuracy: 0.8934 - val_loss: 0.3988
Epoch 13/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9727 - loss:
0.0780 - val_accuracy: 0.9035 - val_loss: 0.2954
Epoch 14/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9693 - loss:
0.0910 - val_accuracy: 0.8911 - val_loss: 0.3623
Epoch 15/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9729 - loss:
0.0820 - val_accuracy: 0.9024 - val_loss: 0.3131
Epoch 16/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9694 - loss:
```

```
0.0906 - val_accuracy: 0.9226 - val_loss: 0.2380
Epoch 17/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9777 - loss:
0.0599 - val_accuracy: 0.9293 - val_loss: 0.2434
Epoch 18/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9814 - loss:
0.0702 - val_accuracy: 0.9394 - val_loss: 0.2136
Epoch 19/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9874 - loss:
0.0383 - val_accuracy: 0.9495 - val_loss: 0.1948
Epoch 20/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9809 - loss:
0.0525 - val_accuracy: 0.9158 - val_loss: 0.3709
Epoch 21/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9780 - loss:
0.0626 - val_accuracy: 0.9383 - val_loss: 0.2223
Epoch 22/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9873 - loss:
0.0447 - val_accuracy: 0.9046 - val_loss: 0.4319
Epoch 23/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9904 - loss:
0.0274 - val_accuracy: 0.9315 - val_loss: 0.2437
Epoch 24/50
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9900 - loss:
0.0384 - val_accuracy: 0.8788 - val_loss: 0.5692
```


```
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step
```




Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| No tumor | 1.00 | 0.98 | 0.99 | 175 |
| Tumor | 0.98 | 1.00 | 0.99 | 172 |
| benign | 0.86 | 0.91 | 0.88 | 172 |
| malignant | 0.94 | 0.87 | 0.90 | 186 |
| normal | 0.97 | 0.99 | 0.98 | 186 |
| accuracy | | | 0.95 | 891 |
| macro avg | 0.95 | 0.95 | 0.95 | 891 |
| weighted avg | 0.95 | 0.95 | 0.95 | 891 |

Validation Accuracy: 0.9495