In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc, f1_score, recall
import os
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
import seaborn as sns

# Define tumor classes
path = os.listdir('brain_tumor/Training/')
classes = {'no_tumor': 0, 'pituitary_tumor': 1}

# Load and preprocess data
X = []
Y = []
for cls in classes:
    pth = 'brain_tumor/Training/' + cls
    for j in os.listdir(pth):
        # Load image in grayscale for CNN
        img = cv2.imread(pth + '/' + j, 0)
        img = cv2.resize(img, (200, 200))
        img_filtered = cv2.GaussianBlur(img, (5, 5), 0)  # Gaussian blur
        X.append(img_filtered)
        Y.append(classes[cls])

X = np.array(X)
Y = np.array(Y)

# Reshape and normalize data for CNN
X_cnn = X.reshape(len(X), 200, 200, 1)  # Reshape for CNN input
X_cnn = X_cnn / 255.0  # Normalize pixel values
Y_cnn = tf.keras.utils.to_categorical(Y, num_classes=2)  # One-hot encode la

# Split the dataset for CNN
xtrain, xtest, ytrain, ytest = train_test_split(X_cnn, Y_cnn, random_state=1

# 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'
)

# Fit the data generator to the training data
datagen.fit(xtrain)
```

```python
# Build Enhanced CNN model with Regularization
def build_cnn_model(input_shape):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer=regu
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', kernel_regularizer=reg
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(256, (3, 3), activation='relu', kernel_regularizer=reg
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu', kernel_regularizer=regularizers
        layers.Dropout(0.5),
        layers.Dense(256, activation='relu', kernel_regularizer=regularizers
        layers.Dropout(0.5),
        layers.Dense(2, activation='softmax')
    ])
    return model

# Initialize CNN model
input_shape = (200, 200, 1)
cnn_model = build_cnn_model(input_shape)

# Compile CNN model
optimizer = Adam(learning_rate=0.0001)
cnn_model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

# Train CNN model with data augmentation
history = cnn_model.fit(datagen.flow(xtrain, ytrain, batch_size=32),
                        epochs=20,
                        validation_data=(xtest, ytest))

# Evaluate CNN model
test_loss, test_acc = cnn_model.evaluate(xtest, ytest)
print(f"CNN Test Accuracy: {test_acc:.4f}")

# Plot training and validation accuracy and loss
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.title('Training and Validation Accuracy', fontsize=16)
plt.legend()
plt.grid(True, alpha=0.3)

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
```

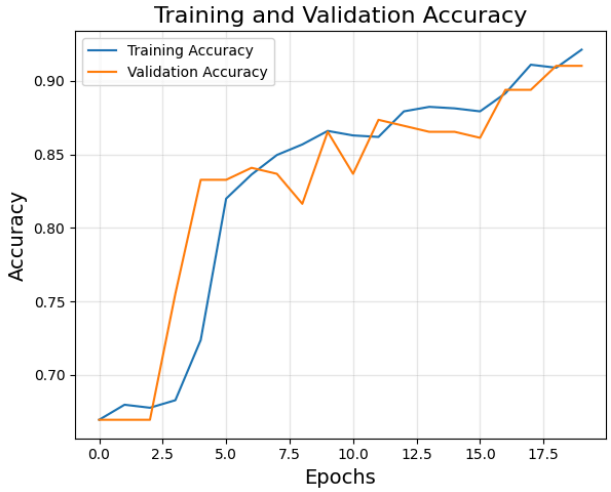
```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs', fontsize=14)
plt.ylabel('Loss', fontsize=14)
plt.title('Training and Validation Loss', fontsize=16)
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# ROC Curve for CNN
cnn_probs = cnn_model.predict(xtest)[:, 1]
fpr_cnn, tpr_cnn, _ = roc_curve(ytest.argmax(axis=1), cnn_probs)
roc_auc_cnn = auc(fpr_cnn, tpr_cnn)

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(ytest.argmax(axis=1), cnn_prob

plt.figure(figsize=(12, 5))

# ROC Curve
plt.subplot(1, 2, 1)
plt.plot(fpr_cnn, tpr_cnn, color='blue', lw=2, label=f'ROC Curve (AUC = {roc
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('ROC Curve', fontsize=16)
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)

# Precision-Recall Curve
plt.subplot(1, 2, 2)
plt.plot(recall, precision, color='green', lw=2, label='Precision-Recall Cur
plt.xlabel('Recall', fontsize=14)
plt.ylabel('Precision', fontsize=14)
plt.title('Precision-Recall Curve', fontsize=16)
plt.legend(loc='lower left')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Get predictions from the CNN model
cnn_preds = cnn_model.predict(xtest)
cnn_preds_classes = np.argmax(cnn_preds, axis=1)
true_labels = np.argmax(ytest, axis=1)

# Calculate confusion matrix
conf_matrix = confusion_matrix(true_labels, cnn_preds_classes)

# Calculate F1 Score, Recall, and Precision
f1 = f1_score(true_labels, cnn_preds_classes)
recall = recall_score(true_labels, cnn_preds_classes)
precision = precision_score(true_labels, cnn_preds_classes)

# Print the metrics
```

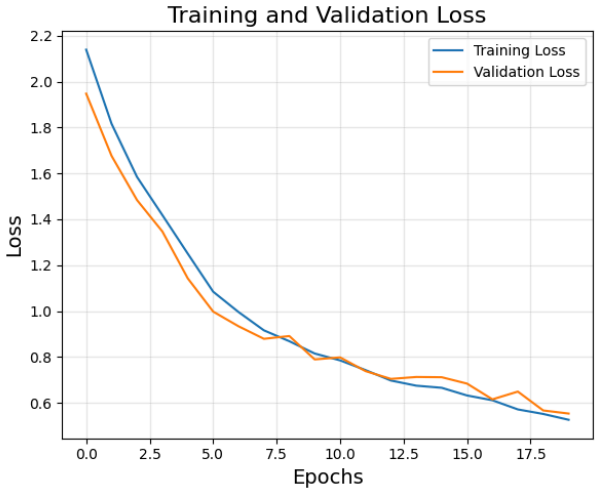
```
print(f"F1 Score: {f1:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Precision: {precision:.4f}")

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True,
            xticklabels=['No Tumor', 'Pituitary Tumor'],
            yticklabels=['No Tumor', 'Pituitary Tumor'],
            annot_kws={"size": 14},
            linewidths=0.5,
            square=True)
plt.xlabel('Predicted Labels', fontsize=14)
plt.ylabel('True Labels', fontsize=14)
plt.title('Confusion Matrix', fontsize=16, pad=20)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

```
C:\Users\Ananda kumar sahoo\AppData\Local\Programs\Python\Python312\Lib\site
-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do n
ot pass an `input_shape`/`input_dim` argument to a layer. When using Sequent
ial models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
C:\Users\Ananda kumar sahoo\AppData\Local\Programs\Python\Python312\Lib\site
-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWa
rning: Your `PyDataset` class should call `super().__init__(**kwargs)` in it
s constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max
_queue_size`. Do not pass these arguments to `fit()`, as they will be ignore
d.
  self._warn_if_super_not_called()
```

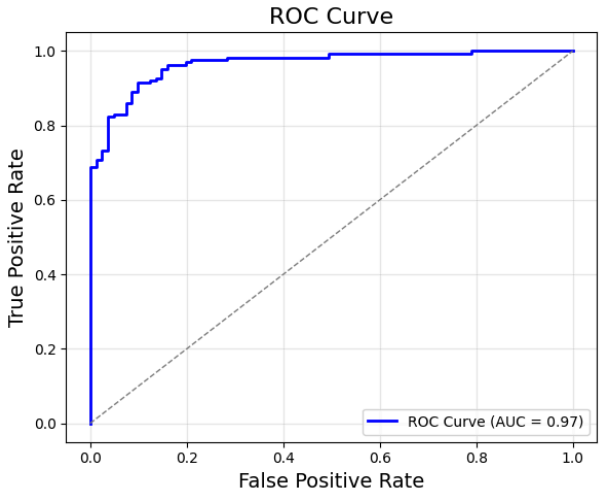
```
Epoch 1/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 28s 839ms/step - accuracy: 0.6505 - loss: 2.2303
- val_accuracy: 0.6694 - val_loss: 1.9477
Epoch 2/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 861ms/step - accuracy: 0.6766 - loss: 1.8827
- val_accuracy: 0.6694 - val_loss: 1.6766
Epoch 3/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 28s 889ms/step - accuracy: 0.6653 - loss: 1.6372
- val_accuracy: 0.6694 - val_loss: 1.4841
Epoch 4/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 878ms/step - accuracy: 0.6979 - loss: 1.4388
- val_accuracy: 0.7551 - val_loss: 1.3471
Epoch 5/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 871ms/step - accuracy: 0.6900 - loss: 1.3005
- val_accuracy: 0.8327 - val_loss: 1.1429
Epoch 6/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 842ms/step - accuracy: 0.8077 - loss: 1.1181
- val_accuracy: 0.8327 - val_loss: 0.9979
Epoch 7/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 854ms/step - accuracy: 0.8478 - loss: 0.9894
- val_accuracy: 0.8408 - val_loss: 0.9339
Epoch 8/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 839ms/step - accuracy: 0.8542 - loss: 0.9270
- val_accuracy: 0.8367 - val_loss: 0.8793
Epoch 9/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 844ms/step - accuracy: 0.8578 - loss: 0.8895
- val_accuracy: 0.8163 - val_loss: 0.8913
Epoch 10/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 858ms/step - accuracy: 0.8589 - loss: 0.8306
- val_accuracy: 0.8653 - val_loss: 0.7894
Epoch 11/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 840ms/step - accuracy: 0.8590 - loss: 0.7874
- val_accuracy: 0.8367 - val_loss: 0.7984
Epoch 12/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 845ms/step - accuracy: 0.8472 - loss: 0.7667
- val_accuracy: 0.8735 - val_loss: 0.7385
Epoch 13/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 845ms/step - accuracy: 0.8772 - loss: 0.7066
- val_accuracy: 0.8694 - val_loss: 0.7046
Epoch 14/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 867ms/step - accuracy: 0.8747 - loss: 0.6870
- val_accuracy: 0.8653 - val_loss: 0.7129
Epoch 15/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 846ms/step - accuracy: 0.8692 - loss: 0.6995
- val_accuracy: 0.8653 - val_loss: 0.7119
Epoch 16/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 863ms/step - accuracy: 0.8858 - loss: 0.6257
- val_accuracy: 0.8612 - val_loss: 0.6841
Epoch 17/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 842ms/step - accuracy: 0.8786 - loss: 0.6420
- val_accuracy: 0.8939 - val_loss: 0.6155
Epoch 18/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 851ms/step - accuracy: 0.9126 - loss: 0.5578
- val_accuracy: 0.8939 - val_loss: 0.6493
Epoch 19/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 26s 849ms/step - accuracy: 0.9142 - loss: 0.5423
```

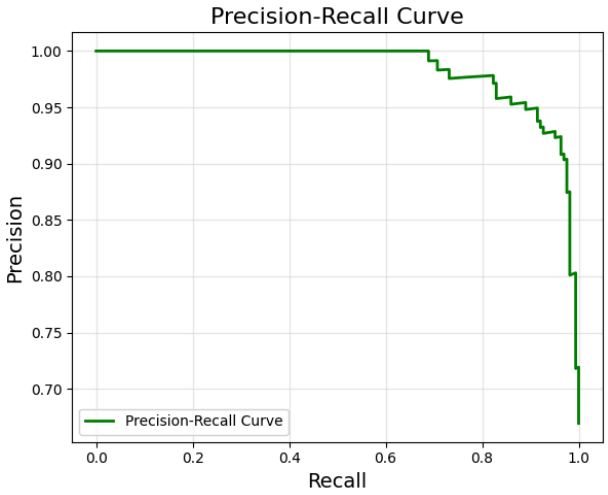
```
- val_accuracy: 0.9102 - val_loss: 0.5670
Epoch 20/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 27s 870ms/step - accuracy: 0.9300 - loss: 0.5167
- val_accuracy: 0.9102 - val_loss: 0.5536
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 158ms/step - accuracy: 0.8965 - loss: 0.5572
CNN Test Accuracy: 0.9102
```



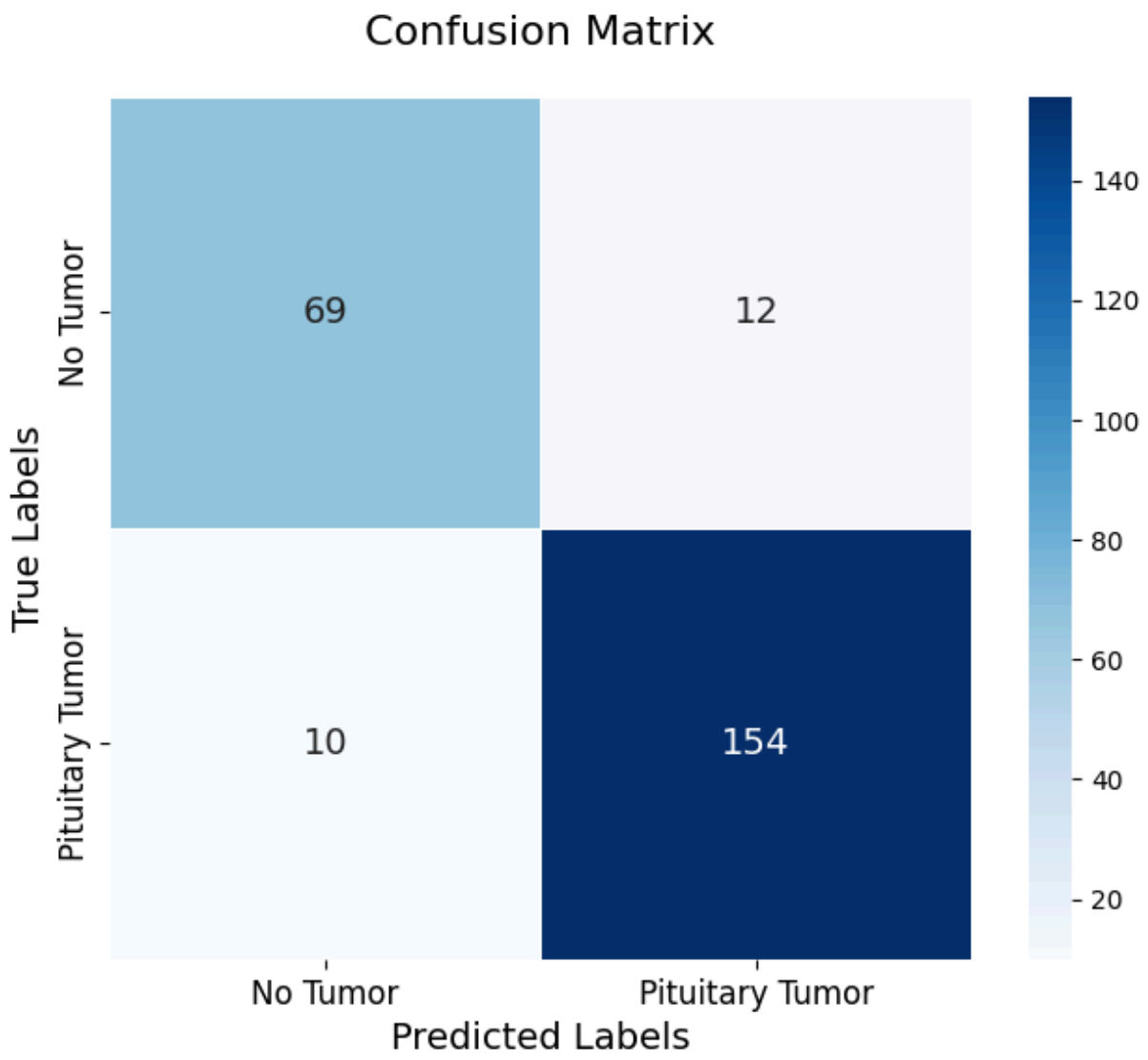



```
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 153ms/step
```






```
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 140ms/step
F1 Score: 0.9333
Recall: 0.9390
Precision: 0.9277
```



In [ ]: