```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Define constants
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 25
NUM_CLASSES = 4

# Data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,
    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'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Load the data
train_generator = train_datagen.flow_from_directory(
    "C:/Vineeth MSc Project/Datasets/Brain Tumor MRI Dataset/Training",
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    "C:/Vineeth MSc Project/Datasets/Brain Tumor MRI Dataset/Testing",
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

# Build VGG16 model
def build_vgg16_model():
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE
    x = Flatten()(base_model.output)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(NUM_CLASSES, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=x)

    for layer in base_model.layers:
        layer.trainable = False

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
# Build custom advanced CNN model
def build_custom_cnn_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(NUM_CLASSES, activation='softmax')
    ])

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

# Train models
vgg16_model = build_vgg16_model()
custom_cnn_model = build_custom_cnn_model()

vgg16_history = vgg16_model.fit(train_generator, epochs=EPOCHS, validation_data=test_generato
custom_cnn_history = custom_cnn_model.fit(train_generator, epochs=EPOCHS, validation_data=tes

# Evaluate models
test_generator.reset()
vgg16_predictions = vgg16_model.predict(test_generator)
custom_cnn_predictions = custom_cnn_model.predict(test_generator)

vgg16_predicted_classes = np.argmax(vgg16_predictions, axis=1)
custom_cnn_predicted_classes = np.argmax(custom_cnn_predictions, axis=1)

true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Classification reports
vgg16_report = classification_report(true_classes, vgg16_predicted_classes, target_names=clas
custom_cnn_report = classification_report(true_classes, custom_cnn_predicted_classes, target_

print("VGG16 Classification Report:\n", vgg16_report)
print("Custom CNN Classification Report:\n", custom_cnn_report)

# Confusion matrices
vgg16_cm = confusion_matrix(true_classes, vgg16_predicted_classes)
custom_cnn_cm = confusion_matrix(true_classes, custom_cnn_predicted_classes)

# Plot confusion matrices
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
sns.heatmap(vgg16_cm, annot=True, fmt='d', cmap='Blues', ax=axes[0], xticklabels=class_labels
axes[0].set_title('VGG16 Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')
```

```
sns.heatmap(custom_cnn_cm, annot=True, fmt='d', cmap='Blues', ax=axes[1], xticklabels=class_l
axes[1].set_title('Custom CNN Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('True')

plt.show()

# Plot accuracy and loss curves
def plot_history(history, model_name):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b', label='Training accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
    plt.title(f'{model_name} Training and Validation Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title(f'{model_name} Training and Validation Loss')
    plt.legend()

    plt.show()

plot_history(vgg16_history, 'VGG16')
plot_history(custom_cnn_history, 'Custom CNN')
```