```
# IMPORTANT: SOME KAGGLE DATA SOURCES ARE PRIVATE
# RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES.
import kagglehub
kagglehub.login()
```

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

aabc1729_braintumor_data_path = kagglehub.dataset_download('aabc1729/braintumor-data')

print('Data source import complete.')
```

Start coding or generate with AI.

```
# Step 1: Import Required Libraries
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.preprocessing import LabelEncoder
```

```
# Step 2: Define the Dataset Directory
extraction_dir = '/kaggle/input/braintumor-data/Braintumor Detection'

# Verify the contents of the directory to ensure the data is there
print("Directory contents:", os.listdir(extraction_dir))

# Categories (subfolders) within the extraction directory
categories = ['glioma', 'notumor', 'meningioma', 'pituitary']
```

```
Directory contents: ['pituitary', 'notumor', 'meningioma', 'glioma']
```

Start coding or generate with AI.

```
import os
import numpy as np
from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import LabelEncoder

# Function to load and preprocess images
def load_and_preprocess_images(base_dir, categories, target_size=(150, 150)):
    images = []
    labels = []

    for category in categories:
        folder_path = os.path.join(base_dir, category)
        image_files = os.listdir(folder_path)

        for img_file in image_files:
            img_full_path = os.path.join(folder_path, img_file)

            # Load the image
            try:
                img = image.load_img(img_full_path, target_size=target_size)
                img_array = image.img_to_array(img)
                img_array /= 255.0  # Normalize pixel values (0-1)
                images.append(img_array)
                labels.append(category)
            except Exception as e:
                print(f"Error loading image {img_full_path}: {e}")

    return np.array(images), np.array(labels)

# Load the images and labels
X, y = load_and_preprocess_images(extraction_dir, categories)

# Print the first few labels to see if they are loaded correctly
print("Labels after loading images:", y)

# Check unique labels before encoding
unique_labels, counts = np.unique(y, return_counts=True)
print("Unique labels and their counts:", dict(zip(unique_labels, counts)))

# Step 4: Encode Labels
label_encoder = LabelEncoder()

# Check the type of y before encoding
print("Type of y before encoding:", type(y))
print("Data type of y elements:", type(y[0]))
```

```
y_encoded = label_encoder.fit_transform(y)

# Verify label encoding
print("Label classes:", label_encoder.classes_)
print("Encoded labels:", y_encoded[:10])  # Show first 10 encoded labels

# Verify unique encoded labels
unique_encoded_labels, encoded_counts = np.unique(y_encoded, return_counts=True)
print("Unique encoded labels and their counts:", dict(zip(unique_encoded_labels, encoded_counts)))
```

```
Labels after loading images: ['glioma' 'glioma' 'glioma' ... 'pituitary' 'pituitary' 'pituitary']
Unique labels and their counts: {'glioma': 1321, 'meningioma': 1339, 'notumor': 1595, 'pituitary': 1457}
Type of y before encoding: <class 'numpy.ndarray'>
Data type of y elements: <class 'numpy.str_'>
Label classes: ['glioma' 'meningioma' 'notumor' 'pituitary']
Encoded labels: [0 0 0 0 0 0 0 0 0 0]
Unique encoded labels and their counts: {0: 1321, 1: 1339, 2: 1595, 3: 1457}
```

Start coding or generate with AI.

```
for category in categories:
    folder_path = os.path.join(extraction_dir, category)

X, y = load_and_preprocess_images(extraction_dir, categories)
print("Labels after loading images:", y)
```

```
Labels after loading images: ['glioma' 'glioma' 'glioma' ... 'pituitary' 'pituitary' 'pituitary']
```

```
unique, counts = np.unique(y, return_counts=True)
print("Class counts:", dict(zip(unique, counts)))
```

```
Class counts: {'glioma': 1321, 'meningioma': 1339, 'notumor': 1595, 'pituitary': 1457}
```

```
# Step 4: Encode Labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Verify label encoding
print("Label classes:", label_encoder.classes_)
print("Encoded labels:", y_encoded[:10])
```

```
Label classes: ['glioma' 'meningioma' 'notumor' 'pituitary']
Encoded labels: [0 0 0 0 0 0 0 0 0 0]
```

```
# Step 5: Define Train-Test Ratios
ratios = [(0.3, 0.7), (0.5, 0.5), (0.7, 0.3), (0.9, 0.1)]
test_accuracies = []
train_accuracies = []
confusion_matrices = []
classification_reports = []
```

```
# Step 6: Create the CNN Model
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dense(len(categories), activation='softmax')
    ])

    # Compile the model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model
```

```
# Step 7: Train and Evaluate the Model
for train_ratio, test_ratio in ratios:
    print(f"\nTrain/Test Split: {int(train_ratio * 100)}% Train / {int(test_ratio * 100)}% Test")

    # Perform train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, train_size=train_ratio, test_size=test_ratio, random_state=42)

    print(f"Training samples: {len(X_train)}")
    print(f"Testing samples: {len(X_test)}")

    # Create and train the model
    model = create_model()
    model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
```

```
    # Evaluate on test set
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f"Test Accuracy: {test_accuracy:.4f}")
    test_accuracies.append(test_accuracy)
```

```
90/90 ━━━━━━━━━━━━━━━━━━━━ 93s 1s/step - accuracy: 0.9940 - loss: 0.0182
Epoch 10/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 92s 1s/step - accuracy: 0.9898 - loss: 0.0309
90/90 ━━━━━━━━━━━━━━━━━━━━ 23s 253ms/step - accuracy: 0.9264 - loss: 0.2791
Test Accuracy: 0.9293

Train/Test Split: 70% Train / 30% Test
Training samples: 3998
Testing samples: 1714
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.5631 - loss: 1.0978
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 129s 1s/step - accuracy: 0.8548 - loss: 0.3954
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 129s 1s/step - accuracy: 0.9024 - loss: 0.2742
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 142s 1s/step - accuracy: 0.9297 - loss: 0.1834
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 129s 1s/step - accuracy: 0.9700 - loss: 0.0933
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 128s 1s/step - accuracy: 0.9802 - loss: 0.0646
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.9859 - loss: 0.0438
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 130s 1s/step - accuracy: 0.9883 - loss: 0.0287
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 142s 1s/step - accuracy: 0.9931 - loss: 0.0214
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.9851 - loss: 0.0384
54/54 ━━━━━━━━━━━━━━━━━━━━ 15s 266ms/step - accuracy: 0.9564 - loss: 0.2549
Test Accuracy: 0.9533

Train/Test Split: 90% Train / 10% Test
Training samples: 5140
Testing samples: 572
Epoch 1/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 171s 1s/step - accuracy: 0.5590 - loss: 1.0758
Epoch 2/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 170s 1s/step - accuracy: 0.8818 - loss: 0.3303
Epoch 3/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 201s 1s/step - accuracy: 0.9330 - loss: 0.1964
Epoch 4/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 226s 1s/step - accuracy: 0.9642 - loss: 0.1065
Epoch 5/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 185s 1s/step - accuracy: 0.9789 - loss: 0.0676
Epoch 6/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 171s 1s/step - accuracy: 0.9877 - loss: 0.0457
Epoch 7/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 191s 1s/step - accuracy: 0.9911 - loss: 0.0303
Epoch 8/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 170s 1s/step - accuracy: 0.9958 - loss: 0.0151
Epoch 9/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 171s 1s/step - accuracy: 0.9935 - loss: 0.0218
Epoch 10/10
161/161 ━━━━━━━━━━━━━━━━━━━━ 173s 1s/step - accuracy: 0.9940 - loss: 0.0174
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 258ms/step - accuracy: 0.9595 - loss: 0.1832
Test Accuracy: 0.9476
```

```
from sklearn.metrics import confusion_matrix, classification_report

# Step 6: Confusion Matrix and Classification Report
y_pred = np.argmax(model.predict(X_test), axis=1)

# Initialize lists to store confusion matrices and classification reports
confusion_matrices = []
classification_reports = []


    # Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
confusion_matrices.append(cm)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 262ms/step
```

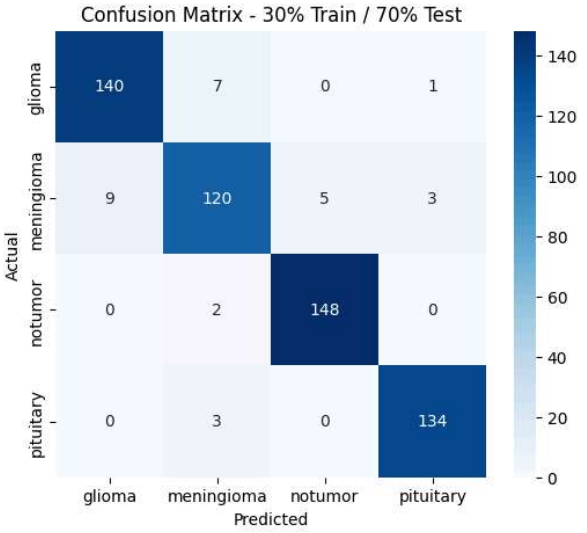
```
for train_ratio, test_ratio in ratios:
    print(f"\nTrain/Test Split: {int(train_ratio * 100)}% Train / {int(test_ratio * 100)}% Test")

# Classification report
    cr = classification_report(y_test, y_pred, target_names=label_encoder.classes_)
    classification_reports.append(cr)  # Append the classification report

    # Print Classification Report with the correct format
    print(f"Classification Report for {int(train_ratio * 100)}% Train:")
    print(cr)
```

```
Train/Test Split: 30% Train / 70% Test
Classification Report for 30% Train:
              precision    recall  f1-score   support

      glioma       0.94      0.95      0.94       148
  meningioma       0.91      0.88      0.89       137
```

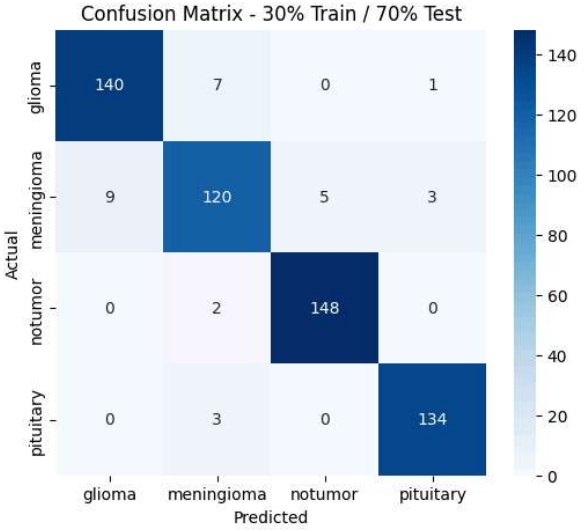
```
           notumor       0.97      0.99      0.98       150
         pituitary       0.97      0.98      0.97       137

          accuracy                           0.95       572
         macro avg       0.95      0.95      0.95       572
      weighted avg       0.95      0.95      0.95       572


      Train/Test Split: 50% Train / 50% Test
      Classification Report for 50% Train:
                    precision    recall  f1-score   support

            glioma       0.94      0.95      0.94       148
        meningioma       0.91      0.88      0.89       137
           notumor       0.97      0.99      0.98       150
         pituitary       0.97      0.98      0.97       137

          accuracy                           0.95       572
         macro avg       0.95      0.95      0.95       572
      weighted avg       0.95      0.95      0.95       572


      Train/Test Split: 70% Train / 30% Test
      Classification Report for 70% Train:
                    precision    recall  f1-score   support

            glioma       0.94      0.95      0.94       148
        meningioma       0.91      0.88      0.89       137
           notumor       0.97      0.99      0.98       150
         pituitary       0.97      0.98      0.97       137

          accuracy                           0.95       572
         macro avg       0.95      0.95      0.95       572
      weighted avg       0.95      0.95      0.95       572


      Train/Test Split: 90% Train / 10% Test
      Classification Report for 90% Train:
                    precision    recall  f1-score   support

            glioma       0.94      0.95      0.94       148
        meningioma       0.91      0.88      0.89       137
           notumor       0.97      0.99      0.98       150
         pituitary       0.97      0.98      0.97       137

          accuracy                           0.95       572
         macro avg       0.95      0.95      0.95       572
      weighted avg       0.95      0.95      0.95       572
```

```
for train_ratio, test_ratio in ratios:
    print(f"\nTrain/Test Split: {int(train_ratio * 100)}% Train / {int(test_ratio * 100)}% Test")

    ratios_percent = [f"{int(train_ratio * 100)}%" for train_ratio, _ in ratios]


    import seaborn as sns
# Plot Confusion Matrix for each ratio

    for i, (train_ratio, cm) in enumerate(zip(ratios_percent, confusion_matrices)):
        plt.figure(figsize=(6, 5))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
        plt.title(f"Confusion Matrix - {train_ratio} Train / {100 - int(train_ratio[:-1])}% Test")
        plt.ylabel('Actual')
        plt.xlabel('Predicted')
        plt.show()
```

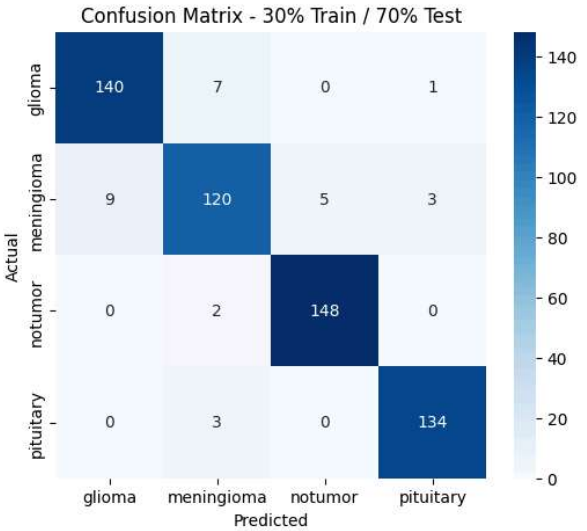
```
Train/Test Split: 30% Train / 70% Test
```



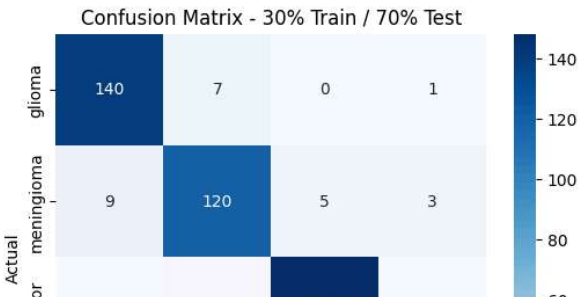

```
Train/Test Split: 50% Train / 50% Test
```

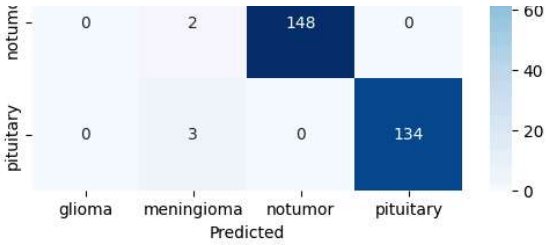



```
Train/Test Split: 70% Train / 30% Test
```




```
Train/Test Split: 90% Train / 10% Test
```






```
for train_ratio, test_ratio in ratios:
    print(f"\nTrain/Test Split: {int(train_ratio * 100)}% Train / {int(test_ratio * 100)}% Test")

    # Perform train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, train_size=train_ratio, test_size=test_ratio, random_state=42)
```

```
Train/Test Split: 30% Train / 70% Test

Train/Test Split: 50% Train / 50% Test

Train/Test Split: 70% Train / 30% Test

Train/Test Split: 90% Train / 10% Test
```

```
train_accuracies = []
test_accuracies = []
confusion_matrices = []
classification_reports = []
```

```
# Define the path where the dataset is stored
extraction_dir = '/kaggle/input/braintumor-data/Braintumor Detection'

# Function to load and preprocess images
def load_and_preprocess_images(base_dir, categories, target_size=(150, 150)):
    images = []
    labels = []

    for category in categories:
        folder_path = os.path.join(base_dir, category)
        image_files = os.listdir(folder_path)

        for img_file in image_files:
            img_full_path = os.path.join(folder_path, img_file)
            img = image.load_img(img_full_path, target_size=target_size)
            img_array = image.img_to_array(img) / 255.0  # Normalize pixel values (0-1)
            images.append(img_array)
            labels.append(category)

    return np.array(images), np.array(labels)

categories = ['glioma', 'notumor', 'meningioma', 'pituitary']
X, y = load_and_preprocess_images(extraction_dir, categories)

# Encode Labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
def create_model(dropout=False, activation='relu', pooling='max'):
    model = Sequential()

    # First Convolutional Layer
    model.add(Conv2D(32, (3, 3), activation=activation, input_shape=(150, 150, 3)))
    if pooling == 'max':
        model.add(MaxPooling2D(pool_size=(2, 2)))
    else:
        model.add(AveragePooling2D(pool_size=(2, 2)))

    # Second Convolutional Layer
    model.add(Conv2D(64, (3, 3), activation=activation))
    if pooling == 'max':
        model.add(MaxPooling2D(pool_size=(2, 2)))
    else:
        model.add(AveragePooling2D(pool_size=(2, 2)))

    # Dropout Layer if specified
    if dropout:
        model.add(Dropout(0.5))

    # Flatten and Output Layer
    model.add(Flatten())
    model.add(Dense(128, activation=activation))
    model.add(Dense(len(categories), activation='softmax'))
```

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

    return model
```

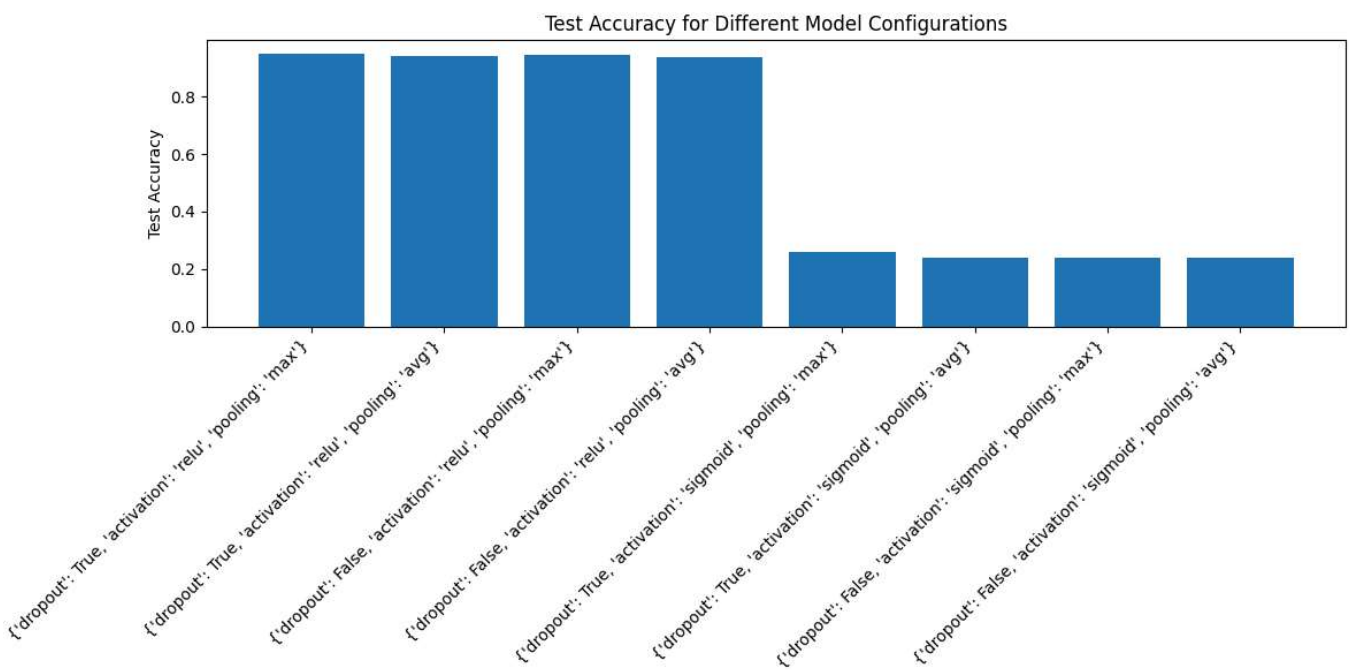
```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, AveragePooling2D
from tensorflow.keras.preprocessing import image
```

```
# Define train/test split
train_ratio = 0.9
test_ratio = 0.1

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, train_size=train_ratio, test_size=test_ratio, random_state=42)

# Configurations to test
configurations = [
    {'dropout': True, 'activation': 'relu', 'pooling': 'max'},
    {'dropout': True, 'activation': 'relu', 'pooling': 'avg'},
    {'dropout': False, 'activation': 'relu', 'pooling': 'max'},
    {'dropout': False, 'activation': 'relu', 'pooling': 'avg'},
    {'dropout': True, 'activation': 'sigmoid', 'pooling': 'max'},
    {'dropout': True, 'activation': 'sigmoid', 'pooling': 'avg'},
    {'dropout': False, 'activation': 'sigmoid', 'pooling': 'max'},
    {'dropout': False, 'activation': 'sigmoid', 'pooling': 'avg'},
]

results = []

for config in configurations:
    print(f"Training with config: {config}")
    model = create_model(dropout=config['dropout'], activation=config['activation'], pooling=config['pooling'])
    model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=1)
    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
    results.append((config, test_accuracy))  # Store results
    print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Training with config: {'dropout': True, 'activation': 'relu', 'pooling': 'max'}
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 107s 654ms/step - accuracy: 0.5909 - loss: 1.0186
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 106s 656ms/step - accuracy: 0.8736 - loss: 0.3340
Epoch 3/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 107s 665ms/step - accuracy: 0.9312 - loss: 0.1836
Epoch 4/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 106s 661ms/step - accuracy: 0.9640 - loss: 0.1059
Epoch 5/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 107s 667ms/step - accuracy: 0.9716 - loss: 0.0734
Test Accuracy: 0.9493
Training with config: {'dropout': True, 'activation': 'relu', 'pooling': 'avg'}
Epoch 1/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 102s 624ms/step - accuracy: 0.5717 - loss: 1.0124
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 143s 628ms/step - accuracy: 0.8717 - loss: 0.3450
Epoch 3/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 101s 628ms/step - accuracy: 0.9215 - loss: 0.2141
Epoch 4/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 142s 626ms/step - accuracy: 0.9499 - loss: 0.1400
Epoch 5/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 102s 630ms/step - accuracy: 0.9643 - loss: 0.1071
Test Accuracy: 0.9406
Training with config: {'dropout': False, 'activation': 'relu', 'pooling': 'max'}
Epoch 1/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 108s 659ms/step - accuracy: 0.6585 - loss: 0.8849
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 106s 658ms/step - accuracy: 0.9199 - loss: 0.2338
Epoch 3/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 105s 652ms/step - accuracy: 0.9645 - loss: 0.1103
Epoch 4/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 143s 657ms/step - accuracy: 0.9767 - loss: 0.0673
Epoch 5/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 142s 656ms/step - accuracy: 0.9874 - loss: 0.0447
Test Accuracy: 0.9441
Training with config: {'dropout': False, 'activation': 'relu', 'pooling': 'avg'}
Epoch 1/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 100s 615ms/step - accuracy: 0.6693 - loss: 0.8075
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 99s 616ms/step - accuracy: 0.9092 - loss: 0.2584
Epoch 3/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 98s 608ms/step - accuracy: 0.9509 - loss: 0.1433
Epoch 4/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 97s 603ms/step - accuracy: 0.9732 - loss: 0.0750
Epoch 5/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 97s 603ms/step - accuracy: 0.9829 - loss: 0.0514
Test Accuracy: 0.9371
Training with config: {'dropout': True, 'activation': 'sigmoid', 'pooling': 'max'}
Epoch 1/5
```

```
161/161 ━━━━━━━━━━━━━━━━━━━━ 116s 711ms/step - accuracy: 0.2526 - loss: 2.3405
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 114s 710ms/step - accuracy: 0.2813 - loss: 1.3835
Epoch 3/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 115s 712ms/step - accuracy: 0.2750 - loss: 1.3872
```

```
# Separate the configurations and their corresponding accuracies
configs, accuracies = zip(*results)

# Plot the accuracies
plt.figure(figsize=(12, 6))
plt.bar(range(len(configs)), accuracies, tick_label=[str(config) for config in configs])
plt.xticks(rotation=45, ha='right')
plt.ylabel('Test Accuracy')
plt.title('Test Accuracy for Different Model Configurations')
plt.tight_layout()
plt.show()
```




Start coding or generate with AI.

Start coding or generate with AI.

```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.preprocessing import LabelEncoder
```

```
extraction_dir = '/kaggle/input/braintumor-data/Braintumor Detection'

# Verify the contents of the directory to ensure the data is there
print("Directory contents:", os.listdir(extraction_dir))

# Categories (subfolders) within the extraction directory
categories = ['glioma', 'notumor', 'meningioma', 'pituitary']
```

```
Directory contents: ['pituitary', 'notumor', 'meningioma', 'glioma']
```

```
def load_and_preprocess_images(base_dir, categories, target_size=(150, 150)):
    images = []
    labels = []

    for category in categories:
        folder_path = os.path.join(base_dir, category)
        image_files = os.listdir(folder_path)

        for img_file in image_files:
            img_full_path = os.path.join(folder_path, img_file)
            img = image.load_img(img_full_path, target_size=target_size)
            img_array = image.img_to_array(img) / 255.0  # Normalize pixel values (0-1)
            images.append(img_array)
            labels.append(category)

    return np.array(images), np.array(labels)
```

```
categories = ['glioma', 'notumor', 'meningioma', 'pituitary']
X, y = load_and_preprocess_images(extraction_dir, categories)

# Encode Labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)



from tensorflow.keras.initializers import HeNormal, GlorotUniform
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import AveragePooling2D, Dropout, Flatten, Dense, Conv2D
from tensorflow.keras.models import Sequential

# Define the model creation function with weight initializer, learning rate, and dropout as parameters
def create_configurable_model(weight_initializer, learning_rate, dropout=True):
    model = Sequential()

    # First Convolutional Layer with configurable initializer
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer=weight_initializer, input_shape=(150, 150, 3)))
    model.add(AveragePooling2D(pool_size=(2, 2)))

    # Second Convolutional Layer
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer=weight_initializer))
    model.add(AveragePooling2D(pool_size=(2, 2)))

    # Third Convolutional Layer
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer=weight_initializer))
    model.add(AveragePooling2D(pool_size=(2, 2)))

    # Flatten the model
    model.add(Flatten())

    # Fully connected Dense layer with optional Dropout
    model.add(Dense(128, activation='relu', kernel_initializer=weight_initializer))
    if dropout:
        model.add(Dropout(0.5))  # Dropout to avoid overfitting

    # Output Layer
    model.add(Dense(len(categories), activation='softmax'))

    # Compile the model with Adam optimizer and the given learning rate
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model




# Configurations to compare
configurations = [
    {'initializer': HeNormal(), 'learning_rate': 0.001},
    {'initializer': GlorotUniform(), 'learning_rate': 0.001},
    {'initializer': HeNormal(), 'learning_rate': 0.0001},
    {'initializer': GlorotUniform(), 'learning_rate': 0.0001},
]

# Iterate through each configuration
for config in configurations:
    print(f"\nTraining with config: Weight Initializer = {config['initializer'].__class__.__name__}, Learning Rate = {config['learning_rate']}")

    # Create the model with the current configuration
    model = create_configurable_model(config['initializer'], config['learning_rate'])

    # Split the dataset (90% train, 10% test)
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, train_size=0.9, test_size=0.1, random_state=42)

    # Train the model
    model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=1)

    # Evaluate the model on the test set
    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Training with config: Weight Initializer = HeNormal, Learning Rate = 0.001
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument t
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 113s 693ms/step - accuracy: 0.6073 - loss: 1.1074
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 113s 699ms/step - accuracy: 0.8564 - loss: 0.4045
Epoch 3/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 111s 691ms/step - accuracy: 0.8997 - loss: 0.2813
Epoch 4/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 111s 691ms/step - accuracy: 0.9167 - loss: 0.2317
Epoch 5/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 111s 688ms/step - accuracy: 0.9385 - loss: 0.1877
Test Accuracy: 0.9336

Training with config: Weight Initializer = GlorotUniform, Learning Rate = 0.001
Epoch 1/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 113s 694ms/step - accuracy: 0.5357 - loss: 1.0414
Epoch 2/5
161/161 ━━━━━━━━━━━━━━━━━━━━ 111s 691ms/step - accuracy: 0.7769 - loss: 0.5601
```