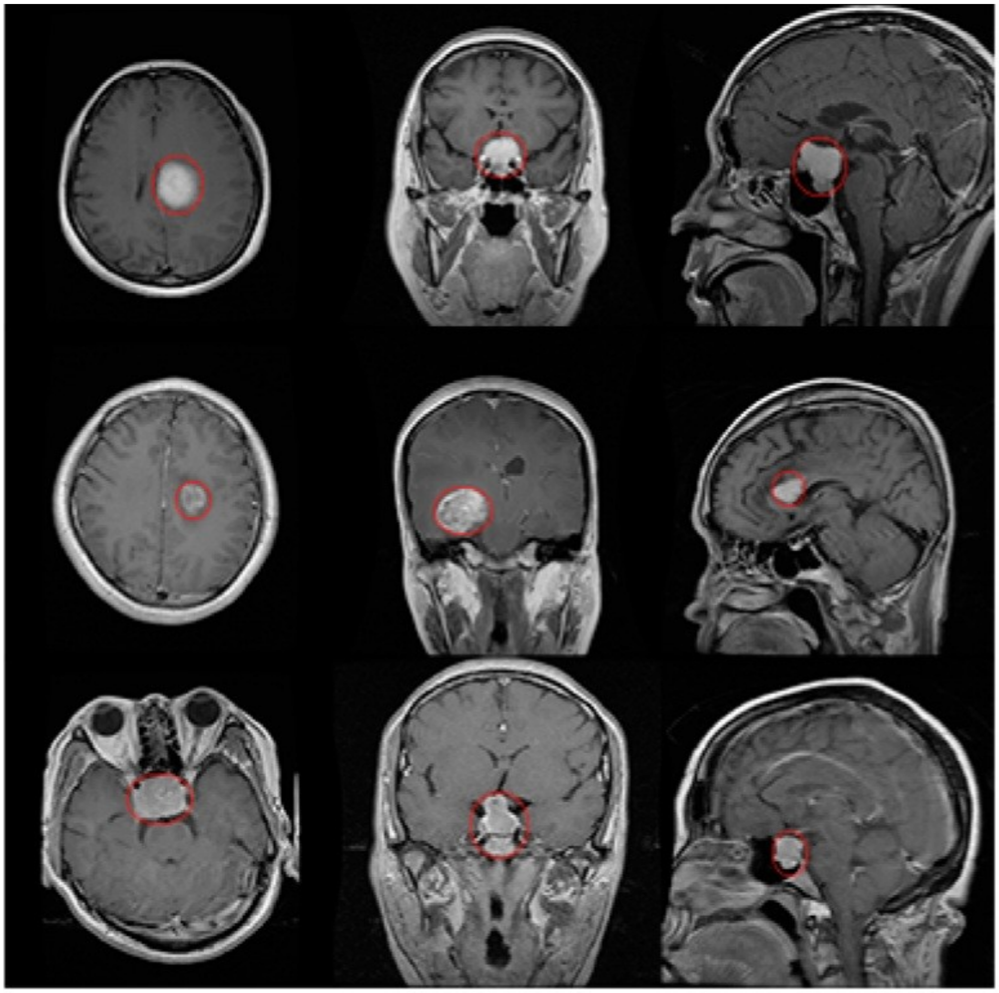
Brain Tumor Classification using FFT and DWT



```
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

base_path = os.path.expanduser('~/Downloads/Data')

data = []

for folder in os.listdir(base_path):
    folder_path = os.path.join(base_path, folder)
```

```
    if os.path.isdir(folder_path):
        for file in os.listdir(folder_path):
            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                image_path = os.path.join(folder_path, file)
                data.append({'image_path': image_path, 'label':
folder})

df = pd.DataFrame(data)

df.head()
```

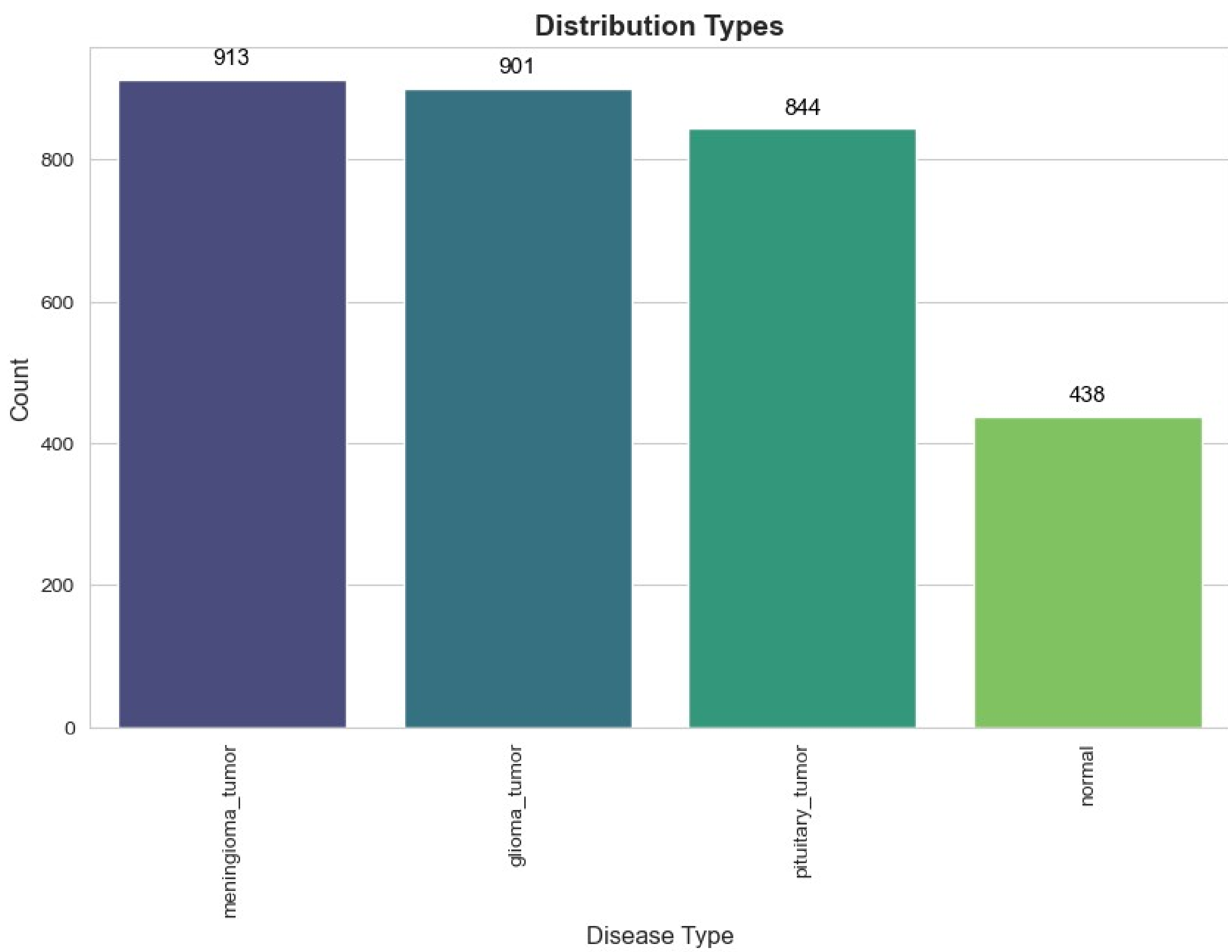
	image_path	label
0	/Users/nirmalgaud/Downloads/Data/meningioma_tu...	meningioma_tumor
1	/Users/nirmalgaud/Downloads/Data/meningioma_tu...	meningioma_tumor
2	/Users/nirmalgaud/Downloads/Data/meningioma_tu...	meningioma_tumor
3	/Users/nirmalgaud/Downloads/Data/meningioma_tu...	meningioma_tumor
4	/Users/nirmalgaud/Downloads/Data/meningioma_tu...	meningioma_tumor

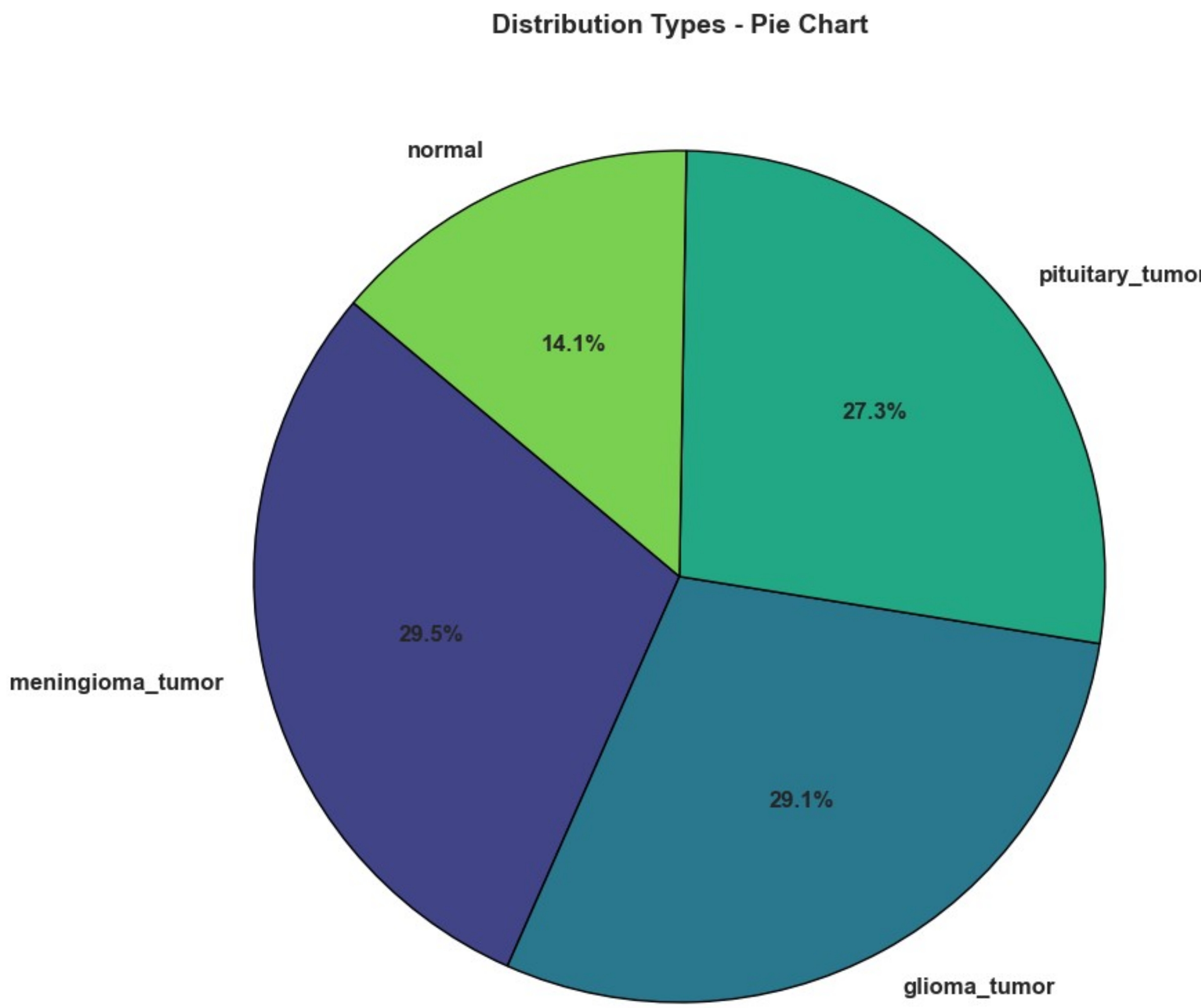
```
df.tail()
```

	image_path	label
3091	/Users/nirmalgaud/Downloads/Data/normal/N_101.jpg	normal
3092	/Users/nirmalgaud/Downloads/Data/normal/N_115.jpg	normal
3093	/Users/nirmalgaud/Downloads/Data/normal/N_288.jpg	normal
3094	/Users/nirmalgaud/Downloads/Data/normal/N_277.jpg	normal
3095	/Users/nirmalgaud/Downloads/Data/normal/N_263.jpg	normal

```
df.shape

(3096, 2)

df.columns

Index(['image_path', 'label'], dtype='object')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3096 entries, 0 to 3095
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  3096 non-null   object
 1   label       3096 non-null   object
dtypes: object(2)
memory usage: 48.5+ KB

df['label'].unique()

array(['meningioma_tumor', 'glioma_tumor', 'pituitary_tumor',
'normal'],
      dtype=object)
```

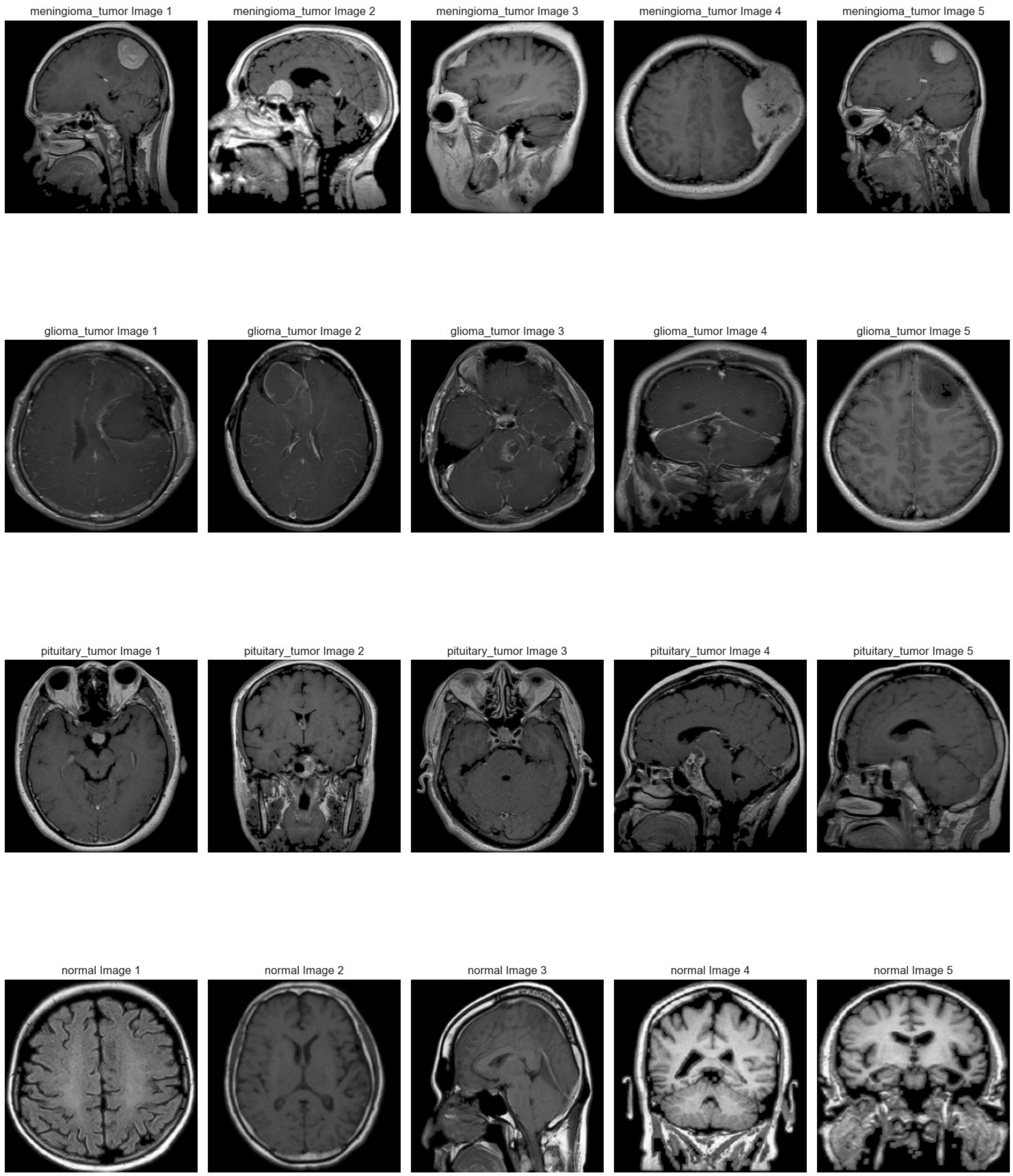
```
df['label'].value_counts()

label
meningioma_tumor    913
glioma_tumor        901
pituitary_tumor     844
normal              438
Name: count, dtype: int64

import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(10, 6))
sns.countplot(data=df, x="label", palette="viridis", ax=ax)

ax.set_title("Distribution Types", fontsize=14, fontweight='bold')
ax.set_xlabel("Disease 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 = 90)
plt.show()

label_counts = df["label"].value_counts()

fig, ax = plt.subplots(figsize=(10, 10))
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 Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution Types
913
901
844
438
800
600
400
200
0
Count
meningioma_tumor
glioma_tumor
pituitary_tumor
normal
Disease Type




```
import cv2

def display_images(df, num_images_per_label=5):

    labels = df['label'].unique()

    fig, axes = plt.subplots(len(labels), num_images_per_label,
figsize=(15, 5 * len(labels)))

    if len(labels) == 1:
        axes = np.array([axes])

    for i, label in enumerate(labels):

        label_df = df[df['label'] ==
```

```
label].sample(n=min(num_images_per_label, len(df[df['label'] ==
label])), random_state=42)

        for j, row in enumerate(label_df.itertuples()):

            img = cv2.imread(row.image_path)
            if img is None:
                print(f"Failed to load image: {row.image_path}")
                continue

            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            ax = axes[i, j] if len(labels) > 1 else axes[j]
            ax.imshow(img)
            ax.set_title(f"{label} Image {j+1}")
            ax.axis('off')

    plt.tight_layout()
    plt.show()

display_images(df, num_images_per_label=5)
```




```python
from sklearn.utils import resample

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

dfs = []
for category in df['label'].unique():
```

```
    class_subset = df[df['label'] == category]
    class_upsampled = resample(class_subset,
                               replace=True,
                               n_samples=max_count,
                               random_state=42)
    dfs.append(class_upsampled)

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

df = df_balanced

df

                                              image_path
label
0     /Users/nirmalgaud/Downloads/Data/glioma_tumor/...
glioma_tumor
1     /Users/nirmalgaud/Downloads/Data/glioma_tumor/...
glioma_tumor
2     /Users/nirmalgaud/Downloads/Data/meningioma_tu...
meningioma_tumor
3      /Users/nirmalgaud/Downloads/Data/normal/N_38.jpg
normal
4     /Users/nirmalgaud/Downloads/Data/pituitary_tum...
pituitary_tumor
...                                                 ...
...
3647  /Users/nirmalgaud/Downloads/Data/glioma_tumor/...
glioma_tumor
3648  /Users/nirmalgaud/Downloads/Data/glioma_tumor/...
glioma_tumor
3649  /Users/nirmalgaud/Downloads/Data/meningioma_tu...
meningioma_tumor
3650  /Users/nirmalgaud/Downloads/Data/normal/N_264.jpg
normal
3651  /Users/nirmalgaud/Downloads/Data/normal/N_205.jpg
normal

[3652 rows x 2 columns]

import cv2
import numpy as np
import pywt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
```

```
import pandas as pd

def clahe_enhance(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        raise ValueError("Image must be 2D (grayscale) or 3D
(BGR/RGB)")
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='haar', level=1):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for cD in coeffs[1]:
        fft_cD = np.fft.fft2(cD)
        features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

class DWTFFTFeatureExtractor(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:
                print(f"Warning: Could not load image from
{image_path}. Skipping.")
                continue
            clahe_img = clahe_enhance(image_array)
            feature_vector = extract_dwt_fft_features(clahe_img)
            features.append(feature_vector)
        return np.array(features)

X = df['image_path'].values
y = df['label'].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

pipeline = Pipeline([
    ('features', DWTFFTFeatureExtractor()),
```

```
    ('scaler', StandardScaler()),
    ('logreg', LogisticRegression(max_iter=1000, solver='liblinear'))
])

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Classification Accuracy: {accuracy}")
```

```
Classification Accuracy: 0.6870437956204379
```

```
import cv2
import numpy as np
import pywt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd
import os

def clahe_enhance(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        raise ValueError("Image must be 2D (grayscale) or 3D
(BGR/RGB)")
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='haar', level=1):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for cD in coeffs[1]:
        fft_cD = np.fft.fft2(cD)
        features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

class DWTFFTFeatureExtractor(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
```

```
        return self
    def transform(self, X):
        features = []
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:

                continue
            clahe_img = clahe_enhance(image_array)
            feature_vector = extract_dwt_fft_features(clahe_img)
            features.append(feature_vector)
        return np.array(features)

X = df['image_path'].values
y = df['label'].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

pipeline = Pipeline([
    ('features', DWTFFTFeatureExtractor()),
    ('scaler', StandardScaler()),
    ('dt_classifier', DecisionTreeClassifier(random_state=42))
])

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Classification Accuracy: {accuracy}")
```

```
Classification Accuracy: 0.8366788321167883
```

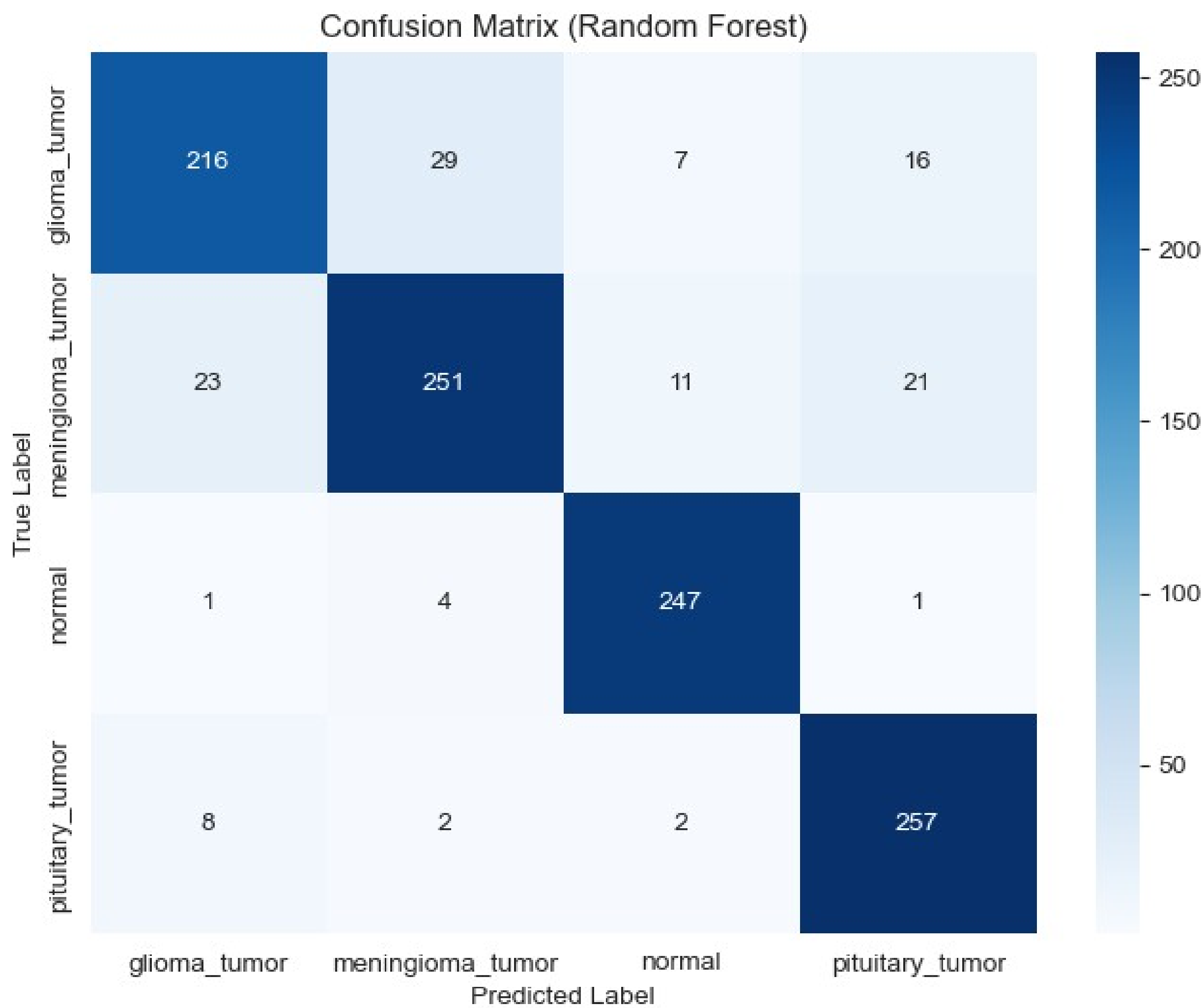
```
import cv2
import numpy as np
import pandas as pd
import pywt
import os
from skimage.feature import graycomatrix, graycoprops
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns

def clahe_enhance(image):
```

```
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        return None
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='haar', level=1):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for cD in coeffs[1]:
        fft_cD = np.fft.fft2(cD)
        features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

def extract_glcm_features(image_clahe):
    image_8bit = (image_clahe / np.max(image_clahe) *
255).astype(np.uint8)
    g = graycomatrix(image_8bit, [1], [0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256, symmetric=True, normed=True)
    properties = ['contrast', 'dissimilarity', 'homogeneity',
'energy', 'correlation', 'ASM']
    glcm_features = [graycoprops(g, prop).ravel() for prop in
properties]
    return np.hstack(glcm_features)

class FeatureLoader(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        processed_images = []
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:
                continue
            clahe_img = clahe_enhance(image_array)
            if clahe_img is not None:
                processed_images.append(clahe_img)
        return np.array(processed_images)

class DWTFFTTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
```

```
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_dwt_fft_features(img))
        return np.array(features)

class GLCMTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_glcm_features(img))
        return np.array(features)

def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix (Random Forest)')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

X = df['image_path'].values
le = LabelEncoder()
y = le.fit_transform(df['label'].values)
num_classes = 4
class_labels = le.classes_

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

y_train = y_train.astype(np.int64)
y_test = y_test.astype(np.int64)

feature_pipeline = Pipeline([
    ('loader', FeatureLoader()),
    ('union', FeatureUnion([
        ('dwt_fft', DWTFFTTransformer()),
        ('glcm', GLCMTransformer())
    ])),
    ('scaler', StandardScaler())
])

X_train_features = feature_pipeline.fit_transform(X_train, y_train)
X_test_features = feature_pipeline.transform(X_test)

rf_model = RandomForestClassifier(n_estimators=100, random_state=42,
```

```python
n_jobs=-1)
rf_model.fit(X_train_features, y_train)

y_pred_test = rf_model.predict(X_test_features)
y_true_test = y_test

accuracy = accuracy_score(y_true_test, y_pred_test)

print("--- Classification Report (Random Forest Model) ---")
print(classification_report(y_true_test, y_pred_test,
target_names=[str(c) for c in class_labels]))
print(f"Overall Accuracy: {accuracy:.4f}")

plot_confusion_matrix(y_true_test, y_pred_test, class_labels)
```

```
--- Classification Report (Random Forest Model) ---
                  precision    recall  f1-score   support

    glioma_tumor       0.87      0.81      0.84       268
meningioma_tumor       0.88      0.82      0.85       306
          normal       0.93      0.98      0.95       253
 pituitary_tumor       0.87      0.96      0.91       269

        accuracy                           0.89      1096
       macro avg       0.89      0.89      0.89      1096
    weighted avg       0.89      0.89      0.88      1096

Overall Accuracy: 0.8859
```




```
import cv2
import numpy as np
import pandas as pd
import pywt
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from skimage.feature import graycomatrix, graycoprops
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
```

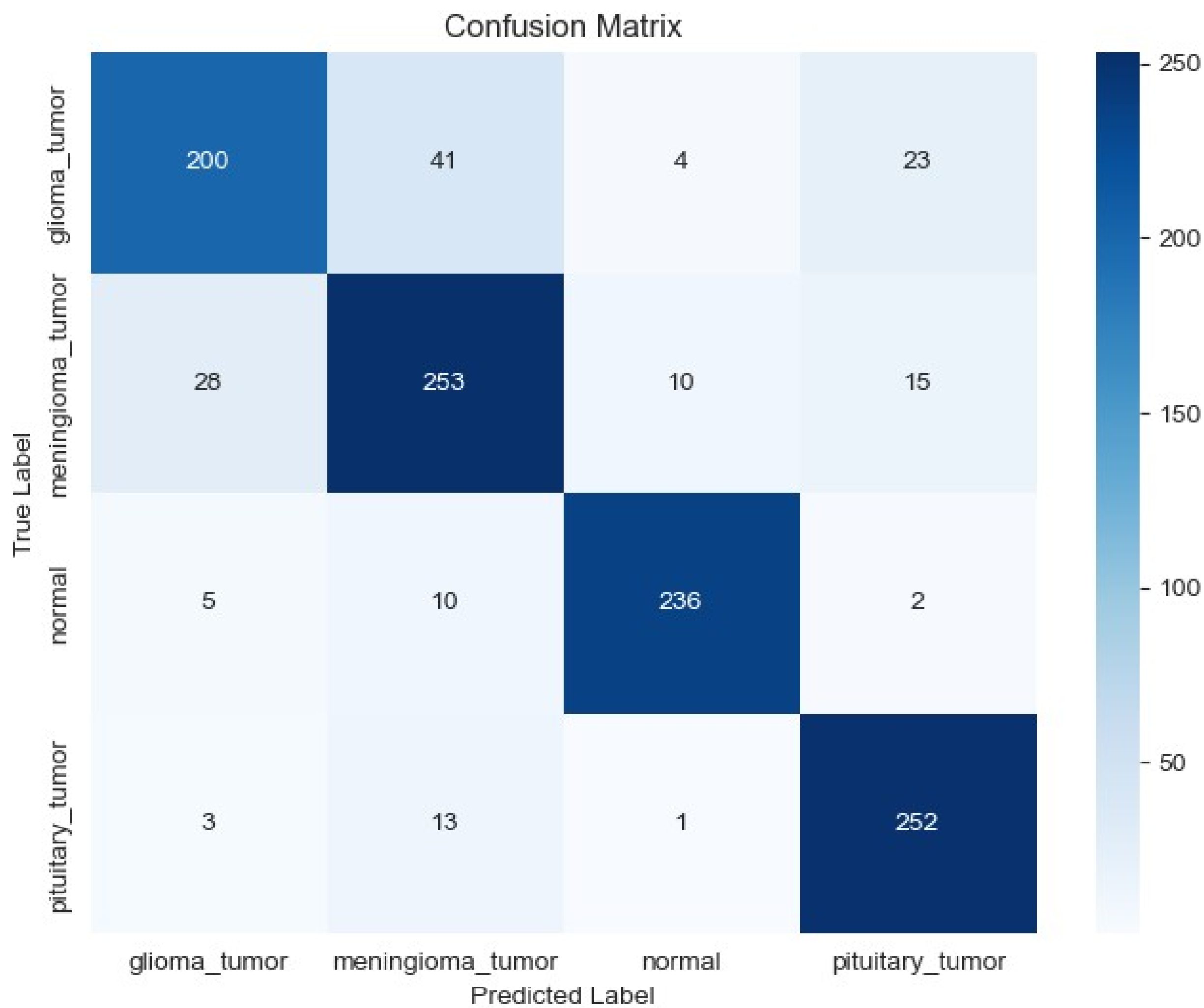
```
def clahe_enhance(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        return None
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='haar', level=1):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for cD in coeffs[1]:
        fft_cD = np.fft.fft2(cD)
        features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

def extract_glcm_features(image_clahe):
    image_8bit = (image_clahe / np.max(image_clahe) *
255).astype(np.uint8)
    g = graycomatrix(image_8bit, [1], [0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256, symmetric=True, normed=True)
    properties = ['contrast', 'dissimilarity', 'homogeneity',
'energy', 'correlation', 'ASM']
    glcm_features = [graycoprops(g, prop).ravel() for prop in
properties]
    return np.hstack(glcm_features)

class FeatureLoader(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        processed_images = []
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:
                continue
            clahe_img = clahe_enhance(image_array)
            if clahe_img is not None:
                processed_images.append(clahe_img)
        return np.array(processed_images)

class DWTFFTTransformer(BaseEstimator, TransformerMixin):
```

```python
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_dwt_fft_features(img))
        return np.array(features)

class GLCMTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_glcm_features(img))
        return np.array(features)

class FeatureDataset(Dataset):
    def __init__(self, features, labels):
        self.features = torch.tensor(features, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

class MultiLayerPerceptron(nn.Module):
    def __init__(self, input_size, num_classes):
        super(MultiLayerPerceptron, self).__init__()
        self.layer1 = nn.Linear(input_size, 256)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(256, 128)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(128, 64)
        self.relu3 = nn.ReLU()
        self.output_layer = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.relu1(self.layer1(x))
        x = self.relu2(self.layer2(x))
        x = self.relu3(self.layer3(x))
        x = self.output_layer(x)
        return x

def train_model(model, train_loader, criterion, optimizer,
num_epochs=50):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
```

```
    model.to(device)
    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

def get_predictions(model, loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    return np.array(all_labels), np.array(all_preds)

def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

X = df['image_path'].values
le = LabelEncoder()
y = le.fit_transform(df['label'].values)
num_classes = 4
class_labels = le.classes_

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

y_train = y_train.astype(np.int64)
y_test = y_test.astype(np.int64)

feature_pipeline = Pipeline([
```

```python
    ('loader', FeatureLoader()),
    ('union', FeatureUnion([
        ('dwt_fft', DWTFFTTransformer()),
        ('glcm', GLCMTransformer())
    ])),
    ('scaler', StandardScaler())
])

X_train_features = feature_pipeline.fit_transform(X_train, y_train)
X_test_features = feature_pipeline.transform(X_test)

input_size = X_train_features.shape[1]
model = MultiLayerPerceptron(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

train_dataset = FeatureDataset(X_train_features, y_train)
test_dataset = FeatureDataset(X_test_features, y_test)
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)

train_model(model, train_loader, criterion, optimizer, num_epochs=50)

y_true_test, y_pred_test = get_predictions(model, test_loader)
accuracy = accuracy_score(y_true_test, y_pred_test)

print("--- Classification Report (PyTorch Model) ---")
print(classification_report(y_true_test, y_pred_test,
target_names=[str(c) for c in class_labels]))
print(f"Overall Accuracy: {accuracy:.4f}")

plot_confusion_matrix(y_true_test, y_pred_test, class_labels)
```

```
--- Classification Report (PyTorch Model) ---
                  precision    recall  f1-score   support

    glioma_tumor       0.85      0.75      0.79       268
meningioma_tumor       0.80      0.83      0.81       306
          normal       0.94      0.93      0.94       253
 pituitary_tumor       0.86      0.94      0.90       269

        accuracy                           0.86      1096
       macro avg       0.86      0.86      0.86      1096
    weighted avg       0.86      0.86      0.86      1096

Overall Accuracy: 0.8586
```




```
import cv2
import numpy as np
import pandas as pd
import pywt
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from skimage.feature import graycomatrix, graycoprops,
local_binary_pattern
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
```

```
import seaborn as sns

def clahe_enhance(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        return None
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='db4', level=2):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for cD in coeffs[1]:
        fft_cD = np.fft.fft2(cD)
        features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

def extract_glcm_features(image_clahe):
    image_8bit = (image_clahe / np.max(image_clahe) *
255).astype(np.uint8)
    g = graycomatrix(image_8bit, [1, 3], [0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256, symmetric=True, normed=True)
    properties = ['contrast', 'dissimilarity', 'homogeneity',
'energy', 'correlation', 'ASM']
    glcm_features = [graycoprops(g, prop).ravel() for prop in
properties]
    return np.hstack(glcm_features)

def extract_lbp_features(image_clahe):
    P = 8
    R = 1
    lbp = local_binary_pattern(image_clahe, P, R, method="uniform")
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp.ravel(), density=True, bins=n_bins,
range=(0, n_bins))
    return hist

class FeatureLoader(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        processed_images = []
```

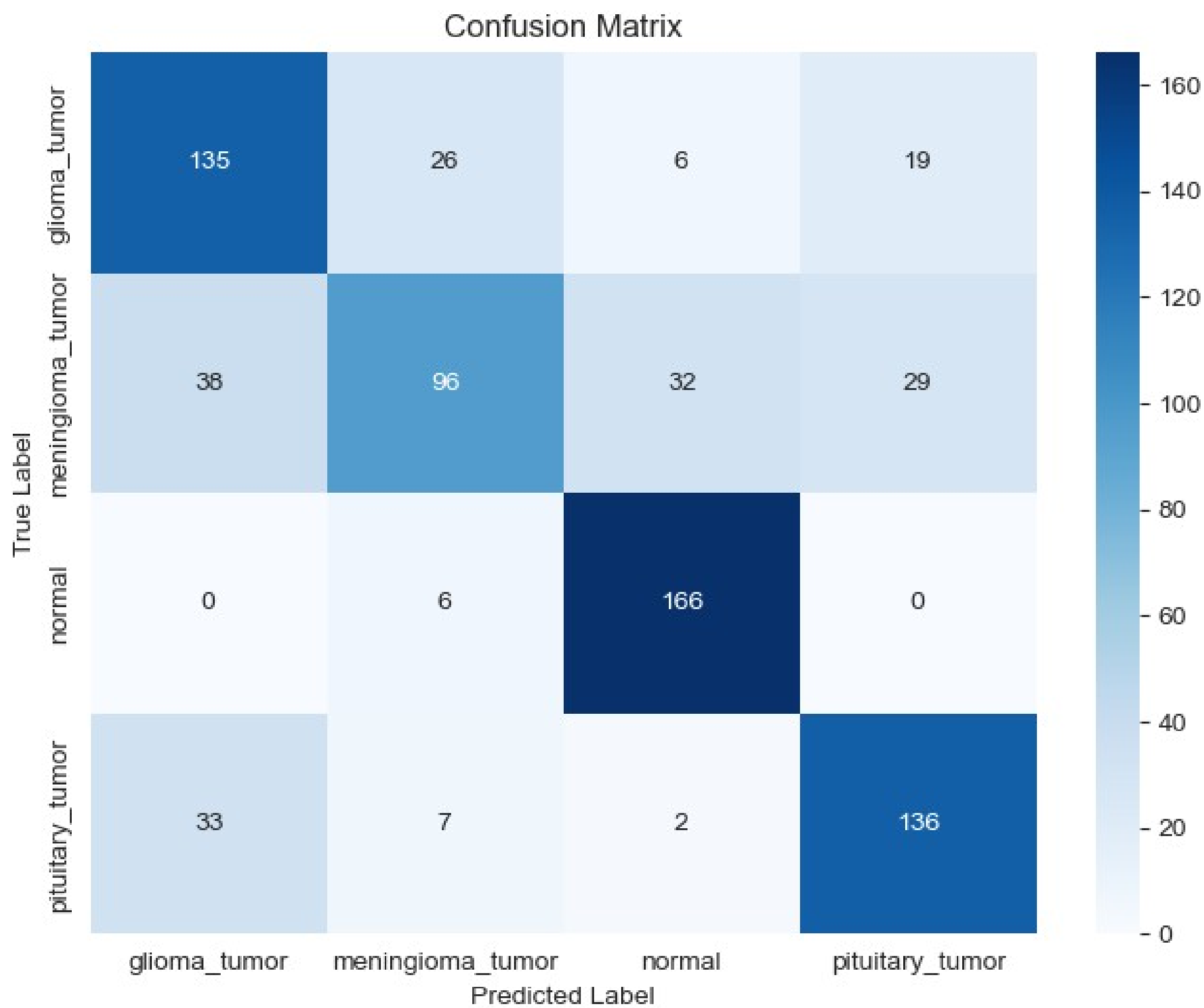
```
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:
                continue
            clahe_img = clahe_enhance(image_array)
            if clahe_img is not None:
                processed_images.append(clahe_img)
        return np.array(processed_images)

class DWTFFTTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_dwt_fft_features(img))
        return np.array(features)

class GLCMTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_glcm_features(img))
        return np.array(features)

class LBPTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_lbp_features(img))
        return np.array(features)

class FeatureDataset(Dataset):
    def __init__(self, features, labels):
        self.features = torch.tensor(features, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

class MultiLayerPerceptron(nn.Module):
    def __init__(self, input_size, num_classes):
        super(MultiLayerPerceptron, self).__init__()
```

```
        self.layer1 = nn.Linear(input_size, 256)
        self.relu1 = nn.ReLU()
        self.drop1 = nn.Dropout(0.3)
        self.layer2 = nn.Linear(256, 128)
        self.relu2 = nn.ReLU()
        self.drop2 = nn.Dropout(0.3)
        self.layer3 = nn.Linear(128, 64)
        self.relu3 = nn.ReLU()
        self.output_layer = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.drop1(self.relu1(self.layer1(x)))
        x = self.drop2(self.relu2(self.layer2(x)))
        x = self.relu3(self.layer3(x))
        x = self.output_layer(x)
        return x

def train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=50, patience=10):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    best_val_loss = float('inf')
    epochs_no_improve = 0

    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

        model.eval()
        val_loss = 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

        avg_val_loss = val_loss / len(val_loader)

        if avg_val_loss < best_val_loss:
            best_val_loss = avg_val_loss
            epochs_no_improve = 0
        else:
```

```
            epochs_no_improve += 1
            if epochs_no_improve == patience:
                break

def get_predictions(model, loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    return np.array(all_labels), np.array(all_preds)

def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

X = df['image_path'].values
le = LabelEncoder()
y = le.fit_transform(df['label'].values)
num_classes = 4
class_labels = le.classes_

X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.15, random_state=42)

y_train = y_train.astype(np.int64)
y_val = y_val.astype(np.int64)
y_test = y_test.astype(np.int64)

feature_pipeline = Pipeline([
    ('loader', FeatureLoader()),
    ('union', FeatureUnion([
        ('dwt_fft', DWTFFTTransformer()),
        ('glcm', GLCMTransformer()),
```

```python
        ('lbp', LBPTransformer())
    ])),
    ('scaler', StandardScaler())
])

X_train_features = feature_pipeline.fit_transform(X_train, y_train)
X_val_features = feature_pipeline.transform(X_val)
X_test_features = feature_pipeline.transform(X_test)

input_size = X_train_features.shape[1]
model = MultiLayerPerceptron(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-
4)

train_dataset = FeatureDataset(X_train_features, y_train)
val_dataset = FeatureDataset(X_val_features, y_val)
test_dataset = FeatureDataset(X_test_features, y_test)

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)

train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=5, patience=15)

y_true_test, y_pred_test = get_predictions(model, test_loader)
accuracy = accuracy_score(y_true_test, y_pred_test)

print("--- Classification Report (PyTorch Model) ---")
print(classification_report(y_true_test, y_pred_test,
target_names=[str(c) for c in class_labels]))
print(f"Overall Accuracy: {accuracy:.4f}")

plot_confusion_matrix(y_true_test, y_pred_test, class_labels)
```

```
--- Classification Report (PyTorch Model) ---
                  precision    recall  f1-score   support

    glioma_tumor       0.66      0.73      0.69       186
meningioma_tumor       0.71      0.49      0.58       195
          normal       0.81      0.97      0.88       172
 pituitary_tumor       0.74      0.76      0.75       178

        accuracy                           0.73       731
       macro avg       0.73      0.74      0.73       731
    weighted avg       0.73      0.73      0.72       731

Overall Accuracy: 0.7291
```




```
import cv2
import numpy as np
import pandas as pd
import pywt
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from skimage.feature import graycomatrix, graycoprops,
local_binary_pattern
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
```

```
import seaborn as sns

def clahe_enhance(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        return None
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='db4', level=2):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for cD in coeffs[1]:
        fft_cD = np.fft.fft2(cD)
        features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

def extract_glcm_features(image_clahe):
    image_8bit = (image_clahe / np.max(image_clahe) *
255).astype(np.uint8)
    g = graycomatrix(image_8bit, [1, 3], [0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256, symmetric=True, normed=True)
    properties = ['contrast', 'dissimilarity', 'homogeneity',
'energy', 'correlation', 'ASM']
    glcm_features = [graycoprops(g, prop).ravel() for prop in
properties]
    return np.hstack(glcm_features)

def extract_lbp_features(image_clahe):
    P = 8
    R = 1
    lbp = local_binary_pattern(image_clahe, P, R, method="uniform")
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp.ravel(), density=True, bins=n_bins,
range=(0, n_bins))
    return hist

class FeatureLoader(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        processed_images = []
```

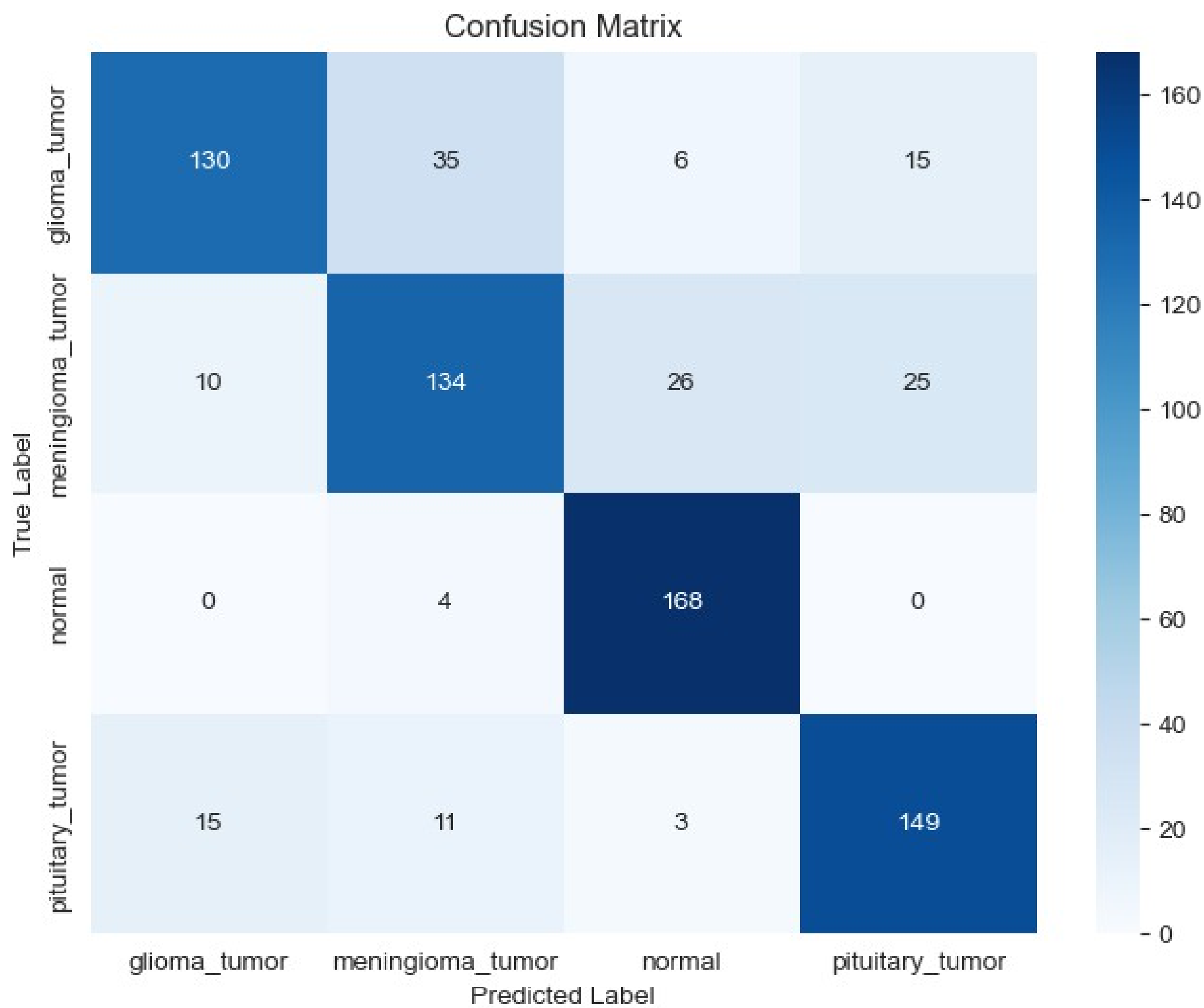
```python
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:
                continue
            clahe_img = clahe_enhance(image_array)
            if clahe_img is not None:
                processed_images.append(clahe_img)
        return np.array(processed_images)

class DWTFFTTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_dwt_fft_features(img))
        return np.array(features)

class GLCMTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_glcm_features(img))
        return np.array(features)

class LBPTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_lbp_features(img))
        return np.array(features)

class FeatureDataset(Dataset):
    def __init__(self, features, labels):
        self.features = torch.tensor(features, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

class MultiLayerPerceptron(nn.Module):
    def __init__(self, input_size, num_classes):
        super(MultiLayerPerceptron, self).__init__()
```

```
        self.layer1 = nn.Linear(input_size, 256)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(256, 128)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(128, 64)
        self.relu3 = nn.ReLU()
        self.output_layer = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.relu1(self.layer1(x))
        x = self.relu2(self.layer2(x))
        x = self.relu3(self.layer3(x))
        x = self.output_layer(x)
        return x

def train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=50, patience=15):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    best_val_loss = float('inf')
    epochs_no_improve = 0

    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

        model.eval()
        val_loss = 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

        avg_val_loss = val_loss / len(val_loader)

        if avg_val_loss < best_val_loss:
            best_val_loss = avg_val_loss
            epochs_no_improve = 0
        else:
            epochs_no_improve += 1
            if epochs_no_improve == patience:
```

```
                break

def get_predictions(model, loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    return np.array(all_labels), np.array(all_preds)

def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

X = df['image_path'].values
le = LabelEncoder()
y = le.fit_transform(df['label'].values)
num_classes = 4
class_labels = le.classes_

X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.15, random_state=42)

y_train = y_train.astype(np.int64)
y_val = y_val.astype(np.int64)
y_test = y_test.astype(np.int64)

feature_pipeline = Pipeline([
    ('loader', FeatureLoader()),
    ('union', FeatureUnion([
        ('dwt_fft', DWTFFTTransformer()),
        ('glcm', GLCMTransformer()),
        ('lbp', LBPTransformer())
    ])),
```

```
    ('scaler', StandardScaler())
])

X_train_features = feature_pipeline.fit_transform(X_train, y_train)
X_val_features = feature_pipeline.transform(X_val)
X_test_features = feature_pipeline.transform(X_test)

input_size = X_train_features.shape[1]
model = MultiLayerPerceptron(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

train_dataset = FeatureDataset(X_train_features, y_train)
val_dataset = FeatureDataset(X_val_features, y_val)
test_dataset = FeatureDataset(X_test_features, y_test)

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)

train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=5, patience=15)

y_true_test, y_pred_test = get_predictions(model, test_loader)
accuracy = accuracy_score(y_true_test, y_pred_test)

print("--- Classification Report (PyTorch Model) ---")
print(classification_report(y_true_test, y_pred_test,
target_names=[str(c) for c in class_labels]))
print(f"Overall Accuracy: {accuracy:.4f}")

plot_confusion_matrix(y_true_test, y_pred_test, class_labels)
```

```
--- Classification Report (PyTorch Model) ---
                  precision    recall  f1-score   support

    glioma_tumor       0.84      0.70      0.76       186
meningioma_tumor       0.73      0.69      0.71       195
          normal       0.83      0.98      0.90       172
 pituitary_tumor       0.79      0.84      0.81       178

        accuracy                           0.79       731
       macro avg       0.80      0.80      0.79       731
    weighted avg       0.79      0.79      0.79       731

Overall Accuracy: 0.7948
```




```
import cv2
import numpy as np
import pandas as pd
import pywt
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from skimage.feature import graycomatrix, graycoprops,
local_binary_pattern
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
```

```
import seaborn as sns

def clahe_enhance(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        gray = image
    else:
        return None
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(gray)

def extract_dwt_fft_features(image_clahe, wavelet='db4', level=2):
    coeffs = pywt.wavedec2(image_clahe, wavelet, level=level)
    features_list = []
    cA = coeffs[0]
    fft_cA = np.fft.fft2(cA)
    features_list.extend([np.mean(np.abs(fft_cA)),
np.std(np.abs(fft_cA)), np.max(np.abs(fft_cA))])
    for level_coeffs in coeffs[1:]:
        for cD in level_coeffs:
            fft_cD = np.fft.fft2(cD)
            features_list.extend([np.mean(np.abs(fft_cD)),
np.std(np.abs(fft_cD)), np.max(np.abs(fft_cD))])
    return np.array(features_list)

def extract_glcm_features(image_clahe):
    image_8bit = (image_clahe / np.max(image_clahe) *
255).astype(np.uint8)
    g = graycomatrix(image_8bit, [1, 3], [0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256, symmetric=True, normed=True)
    properties = ['contrast', 'dissimilarity', 'homogeneity',
'energy', 'correlation', 'ASM']
    glcm_features = [graycoprops(g, prop).ravel() for prop in
properties]
    return np.hstack(glcm_features)

def extract_lbp_features(image_clahe):
    P = 8
    R = 1
    lbp = local_binary_pattern(image_clahe, P, R, method="uniform")
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp.ravel(), density=True, bins=n_bins,
range=(0, n_bins))
    return hist

class FeatureLoader(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
```

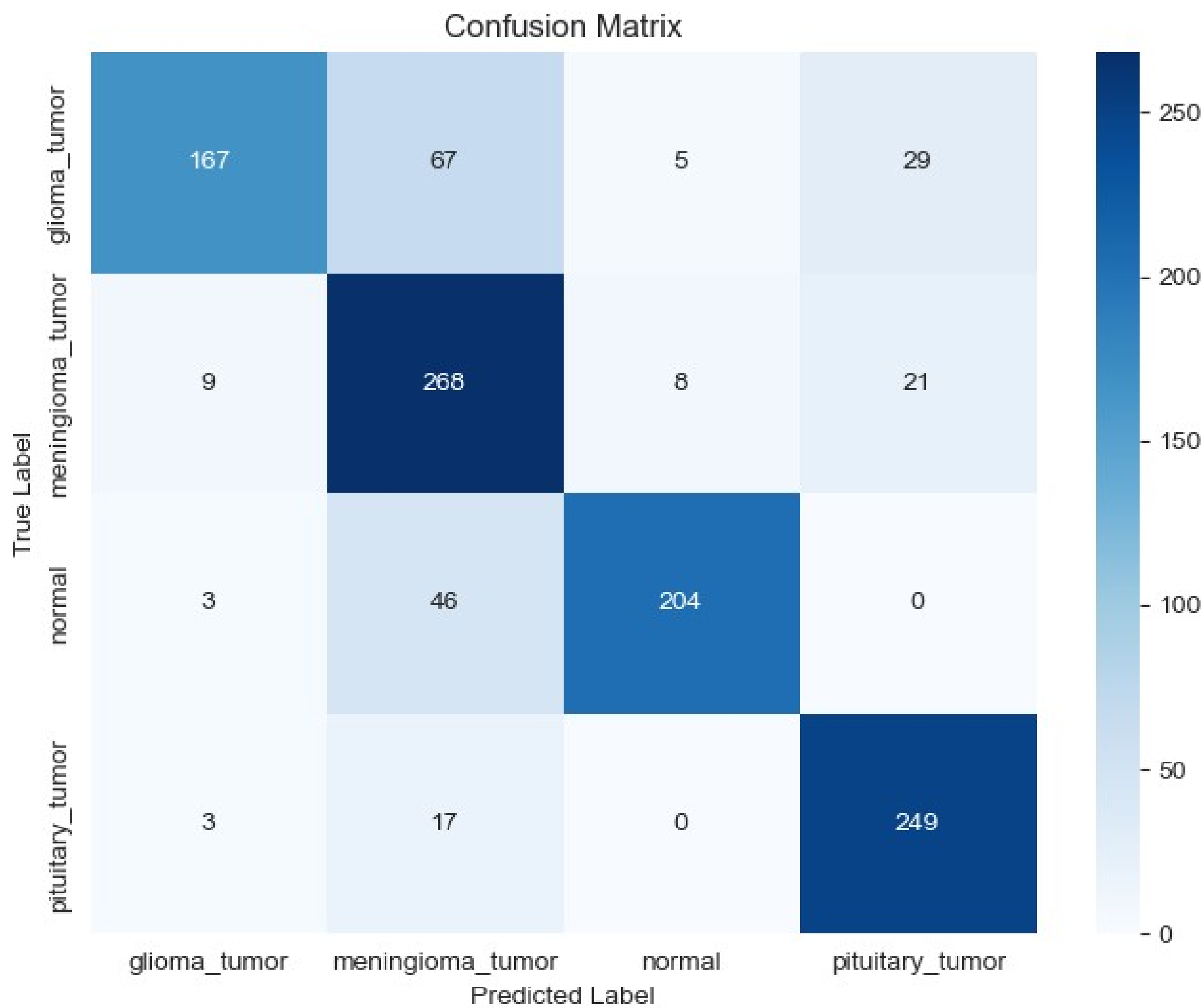
```
        processed_images = []
        for image_path in X:
            image_array = cv2.imread(image_path)
            if image_array is None:
                continue
            clahe_img = clahe_enhance(image_array)
            if clahe_img is not None:
                processed_images.append(clahe_img)
        return np.array(processed_images)

class DWTFFTTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_dwt_fft_features(img))
        return np.array(features)

class GLCMTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_glcm_features(img))
        return np.array(features)

class LBPTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        features = []
        for img in X:
            features.append(extract_lbp_features(img))
        return np.array(features)

class FeatureDataset(Dataset):
    def __init__(self, features, labels):
        self.features = torch.tensor(features, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

class SqueezeExcitationBlock(nn.Module):
    def __init__(self, channel, reduction=4):
```

```
        super(SqueezeExcitationBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool1d(1)
        self.fc = nn.Sequential(
            nn.Linear(channel, channel // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):

        y = self.avg_pool(x.unsqueeze(-1)).squeeze(-1)
        y = self.fc(y)
        return x * y.expand_as(x)

class MultiLayerPerceptron(nn.Module):
    def __init__(self, input_size, num_classes):
        super(MultiLayerPerceptron, self).__init__()
        self.layer1 = nn.Linear(input_size, 256)
        self.relu1 = nn.ReLU()

        self.se_block = SqueezeExcitationBlock(256)

        self.layer2 = nn.Linear(256, 128)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(128, 64)
        self.relu3 = nn.ReLU()
        self.output_layer = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.relu1(self.layer1(x))
        x = self.se_block(x)
        x = self.relu2(self.layer2(x))
        x = self.relu3(self.layer3(x))
        x = self.output_layer(x)
        return x

def train_model(model, train_loader, criterion, optimizer,
num_epochs=50):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
```

```
            optimizer.step()

def get_predictions(model, loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"mps" if torch.backends.mps.is_available() else "cpu")
    model.to(device)
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    return np.array(all_labels), np.array(all_preds)

def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

X = df['image_path'].values
le = LabelEncoder()
y = le.fit_transform(df['label'].values)
num_classes = 4
class_labels = le.classes_

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

y_train = y_train.astype(np.int64)
y_test = y_test.astype(np.int64)

feature_pipeline = Pipeline([
    ('loader', FeatureLoader()),
    ('union', FeatureUnion([
        ('dwt_fft', DWTFFTTransformer()),
        ('glcm', GLCMTransformer()),
        ('lbp', LBPTransformer())
    ])),
    ('scaler', StandardScaler())
])
```

```python
X_train_features = feature_pipeline.fit_transform(X_train, y_train)
X_test_features = feature_pipeline.transform(X_test)

input_size = X_train_features.shape[1]
model = MultiLayerPerceptron(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

train_dataset = FeatureDataset(X_train_features, y_train)
test_dataset = FeatureDataset(X_test_features, y_test)
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)

train_model(model, train_loader, criterion, optimizer, num_epochs=10)

y_true_test, y_pred_test = get_predictions(model, test_loader)
accuracy = accuracy_score(y_true_test, y_pred_test)

print("--- Classification Report (PyTorch Model) ---")
print(classification_report(y_true_test, y_pred_test,
target_names=[str(c) for c in class_labels]))
print(f"Overall Accuracy: {accuracy:.4f}")

plot_confusion_matrix(y_true_test, y_pred_test, class_labels)
```

```
--- Classification Report (PyTorch Model) ---
                  precision    recall  f1-score   support

    glioma_tumor       0.92      0.62      0.74       268
meningioma_tumor       0.67      0.88      0.76       306
          normal       0.94      0.81      0.87       253
 pituitary_tumor       0.83      0.93      0.88       269

        accuracy                           0.81      1096
       macro avg       0.84      0.81      0.81      1096
    weighted avg       0.83      0.81      0.81      1096

Overall Accuracy: 0.8102
```


Confusion Matrix
True Label
glioma_tumor
meningioma_tumor
normal
pituitary_tumor
167
67
5
29
9
268
8
21
3
46
204
0
3
17
0
249
glioma_tumor
meningioma_tumor
normal
pituitary_tumor
Predicted Label
250
200
150
100
50
0