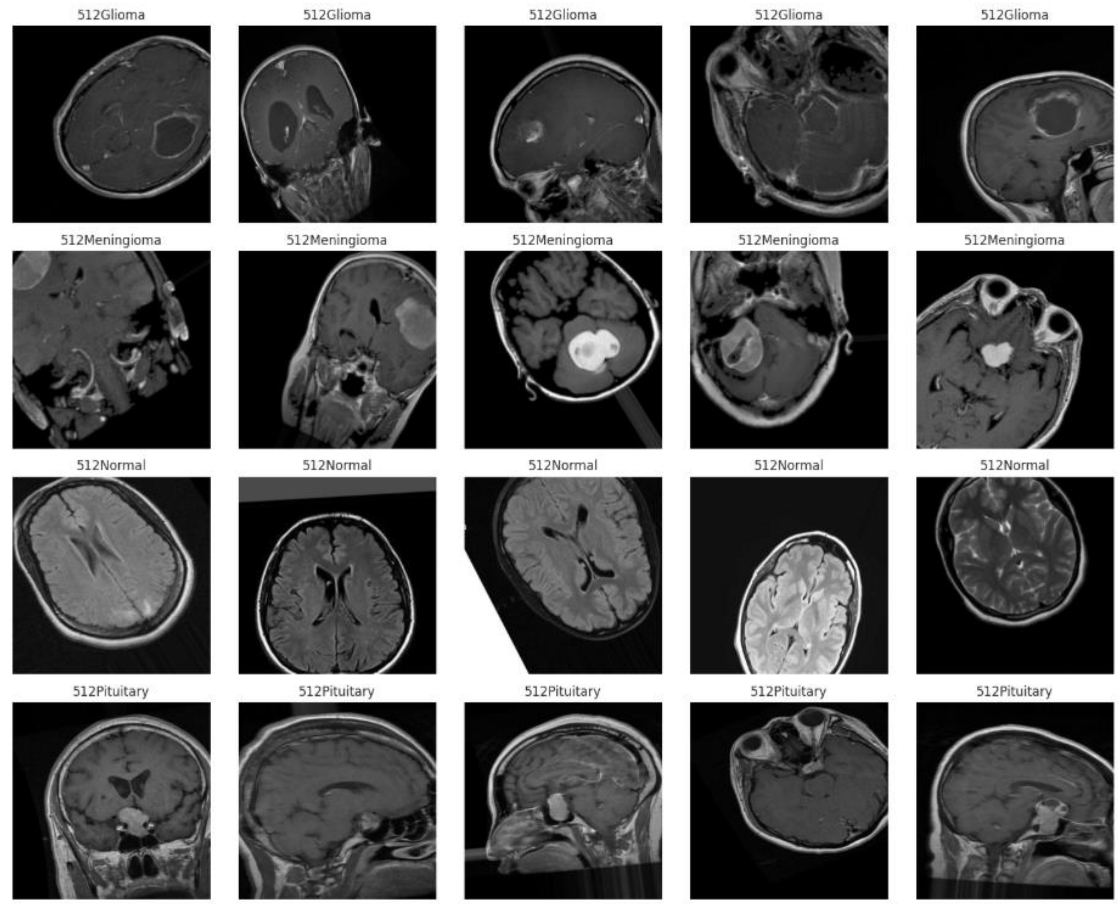
Brain Tumor Multi-Classification with PSO




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

base_path = "/kaggle/input/pmram-bangladeshi-brain-cancer-mri-dataset/PMRAM Bangladeshi Brain Cancer - MRI Dataset/PMRAM Bangladeshi Brain Cancer - MRI Dataset/Augmented Data/Augmented"
categories = ["512Glioma","512Meningioma","512Normal","512Pituitary" ]

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)
```

```python
df = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})
```

```python
df.head()
```

```
                                          image_path       label
0  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Glioma
1  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Glioma
2  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Glioma
3  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Glioma
4  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Glioma
```

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

```
                                             image_path          label
5999  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Pituitary
6000  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Pituitary
6001  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Pituitary
6002  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Pituitary
6003  /kaggle/input/pmram-bangladeshi-brain-cancer-m...  512Pituitary
```

```python
df.shape
```

```
(6004, 2)
```

```python
df.columns
```

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

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

```
0
```

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

```
image_path    0
label         0
dtype: int64
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6004 entries, 0 to 6003
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  6004 non-null   object
 1   label       6004 non-null   object
dtypes: object(2)
memory usage: 93.9+ KB
```

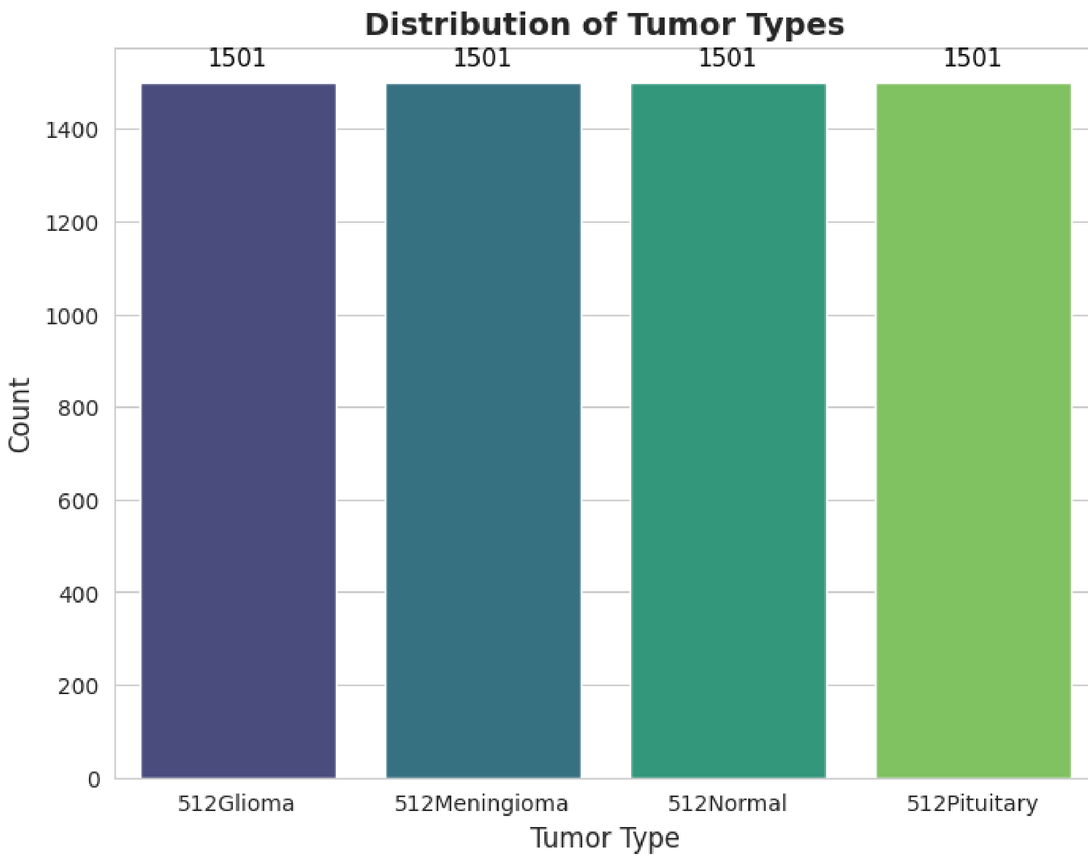
```
df['label'].unique()
```

```
array(['512Glioma', '512Meningioma', '512Normal', '512Pituitary'],
      dtype=object)
```

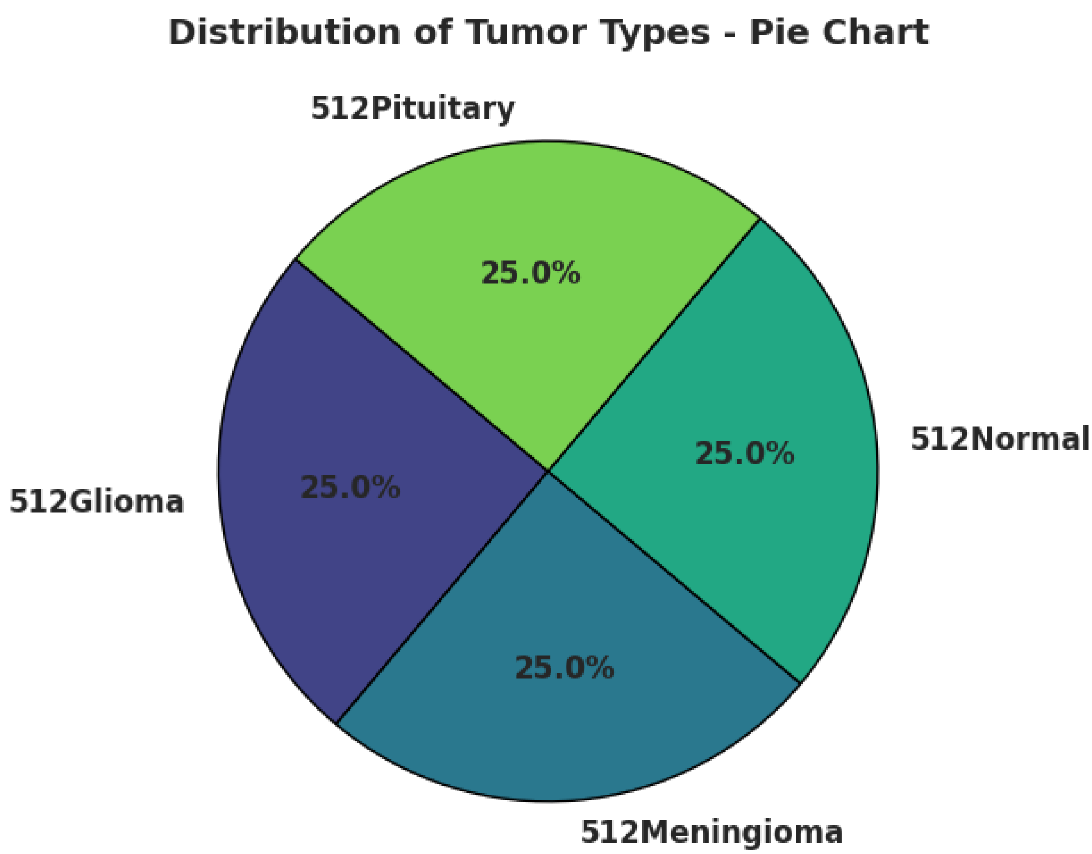
```
df['label'].value_counts()
```

```
label
512Glioma        1501
512Meningioma    1501
512Normal        1501
512Pituitary     1501
Name: count, dtype: int64
```

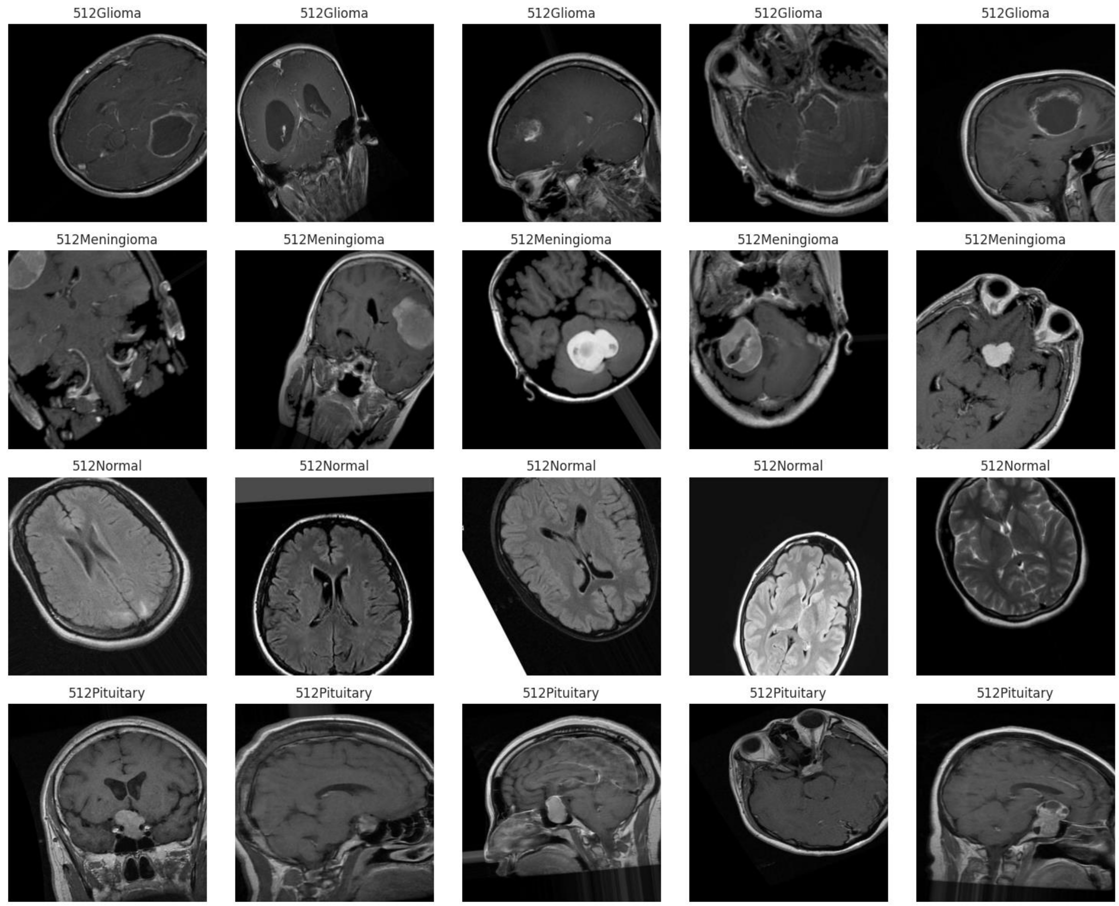
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df, x="label", palette="viridis", ax=ax)

ax.set_title("Distribution of Tumor Types", fontsize=14, fontweight='bold')
ax.set_xlabel("Tumor Type", fontsize=12)
ax.set_ylabel("Count", fontsize=12)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=11, color='black',
                xytext=(0, 5), textcoords='offset points')

plt.show()

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

fig, ax = plt.subplots(figsize=(8, 6))
colors = sns.color_palette("viridis", len(label_counts))

ax.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
       startangle=140, colors=colors, textprops={'fontsize': 12, 'weight':
'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})

ax.set_title("Distribution of Tumor Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```



Distribution of Tumor Types
1501
1501
1501
1501
0
200
400
600
800
1000
1200
1400
Count
512Glioma
512Meningioma
512Normal
512Pituitary
Tumor Type




```
import cv2

num_images = 5

plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_images = df[df['label'] ==
category]['image_path'].iloc[:num_images]

    for j, img_path in enumerate(category_images):

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

        plt.subplot(len(categories), num_images, i * num_images + j + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(category)

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




```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

import warnings
warnings.filterwarnings("ignore")

print ('check')
```

```
check
```

```
train_df_new, temp_df_new = train_test_split(
    df,
    train_size=0.8,
    shuffle=True,
```

```python
    random_state=42,
    stratify=df['label']
)

valid_df_new, test_df_new = train_test_split(
    temp_df_new,
    test_size=0.5,
    shuffle=True,
    random_state=42,
    stratify=temp_df_new['label']
)

batch_size = 16
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

tr_gen = ImageDataGenerator(rescale=1./255)
ts_gen = ImageDataGenerator(rescale=1./255)

train_gen_new = tr_gen.flow_from_dataframe(
    train_df_new,
    x_col='image_path',
    y_col='label',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

valid_gen_new = ts_gen.flow_from_dataframe(
    valid_df_new,
    x_col='image_path',
    y_col='label',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

test_gen_new = ts_gen.flow_from_dataframe(
    test_df_new,
    x_col='image_path',
    y_col='label',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=False,
```

```
    batch_size=batch_size
)
```

```
Found 4803 validated image filenames belonging to 4 classes.
Found 600 validated image filenames belonging to 4 classes.
Found 601 validated image filenames belonging to 4 classes.
```

```
import tensorflow as tf

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  2
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print("GPU is set for TensorFlow")
    except RuntimeError as e:
        print(e)
```

```
GPU is set for TensorFlow
```

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
GlobalAveragePooling2D, Dense, BatchNormalization
from tensorflow.keras import backend as K
import numpy as np
from tensorflow.keras.losses import SparseCategoricalCrossentropy

IMG_WIDTH = 224
IMG_HEIGHT = 224
IMG_CHANNELS = 3
NUM_CLASSES = 4
NUM_PARTICLES = 5
EPOCHS = 3
BATCH_SIZE = 16
W = 0.5
C1 = 1
C2 = 1
LEARNING_RATE_ADAM = 1e-4

def build_classifier(input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)):
    inputs = Input(shape=input_shape)

    c1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    c1 = BatchNormalization()(c1)
    p1 = MaxPooling2D((2, 2))(c1)
```

```python
    c2 = Conv2D(64, (3, 3), activation='relu', padding='same')(p1)
    c2 = BatchNormalization()(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(128, (3, 3), activation='relu', padding='same')(p2)
    c3 = BatchNormalization()(c3)
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(256, (3, 3), activation='relu', padding='same')(p3)
    c4 = BatchNormalization()(c4)
    p4 = MaxPooling2D((2, 2))(c4)

    c5 = Conv2D(512, (3, 3), activation='relu', padding='same')(p4)
    c5 = BatchNormalization()(c5)

    gap = GlobalAveragePooling2D()(c5)
    d1 = Dense(512, activation='relu')(gap)
    d1 = BatchNormalization()(d1)
    outputs = Dense(NUM_CLASSES, activation='softmax')(d1)

    return Model(inputs=inputs, outputs=outputs)

def get_flattened_weights(model):
    weights = model.get_weights()
    flattened = np.concatenate([w.flatten() for w in weights])
    return flattened

def set_weights_from_flat(model, flat_weights):
    weights = []
    index = 0
    for w in model.get_weights():
        shape = w.shape
        size = np.prod(shape)
        weights.append(flat_weights[index:index+size].reshape(shape))
        index += size
    model.set_weights(weights)

scce = SparseCategoricalCrossentropy()

def calculate_fitness(model, flat_weights, generator, loss_function):
    set_weights_from_flat(model, flat_weights)
    total_loss = 0
    total_samples = 0
    for i in range(len(generator)):
        images, labels = generator[i]
        preds = model(images, training=False)
        loss = loss_function(labels, preds)
        total_loss += loss.numpy() * images.shape[0]
        total_samples += images.shape[0]
```

```
    return total_loss / total_samples

def train_classifier_with_pso(model, train_gen, val_gen, loss_function,
                            num_particles=NUM_PARTICLES, epochs=EPOCHS,
                            w=W, c1=C1, c2=C2):

    initial_weights = get_flattened_weights(model)
    dim = len(initial_weights)

    particles = np.array([initial_weights + np.random.randn(dim)*0.1 for _ in
range(num_particles)])
    velocities = np.zeros((num_particles, dim))

    personal_bests = particles.copy()
    personal_best_fitness = np.array([float('inf')] * num_particles)
    global_best = None
    global_best_fitness = float('inf')

    for epoch in range(epochs):
        print(f"PSO Epoch {epoch+1}/{epochs}")

        for i in range(num_particles):
            current_fitness = calculate_fitness(model, particles[i],
train_gen, loss_function)

            if current_fitness < personal_best_fitness[i]:
                personal_best_fitness[i] = current_fitness
                personal_bests[i] = particles[i].copy()

                if current_fitness < global_best_fitness:
                    global_best_fitness = current_fitness
                    global_best = particles[i].copy()

        for i in range(num_particles):
            r1 = np.random.rand(dim)
            r2 = np.random.rand(dim)
            velocities[i] = (w * velocities[i]
                            + c1 * r1 * (personal_bests[i] - particles[i])
                            + c2 * r2 * (global_best - particles[i]))
            particles[i] += velocities[i]

        print(f"Current Best Fitness: {global_best_fitness:.4f}")

    set_weights_from_flat(model, global_best)
    return model

if __name__ == '__main__':
    print("Building classifier model...")
    model_pso = build_classifier()
```

```
    print("Training classifier with PSO...")
    try:
        trained_model_pso = train_classifier_with_pso(
            model_pso,
            train_gen_new,
            valid_gen_new,
            scce
        )
        trained_model_pso.summary()

        print("\n--- Training with Adam for Comparison ---")
        model_adam = build_classifier()
        model_adam.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE_ADAM),
            loss=SparseCategoricalCrossentropy(),
            metrics=['accuracy']
        )

        history_adam = model_adam.fit(
            train_gen_new,
            validation_data=valid_gen_new,
            epochs=EPOCHS,
            verbose=1
        )
        model_adam.summary()

    except ValueError as e:
        print(f"An error occurred during training: {e}")
```

```
Building classifier model...
Training classifier with PSO...
PSO Epoch 1/3
Current Best Fitness: 9.2883
PSO Epoch 2/3
Current Best Fitness: 3.0716
PSO Epoch 3/3
Current Best Fitness: 3.0716

Model: "functional"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━
━━━━━━━━━━┓
┃ Layer (type)                           ┃ Output Shape                   ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━
━━━━━━━━━━┩
│ input_layer (InputLayer)               │ (None, 224, 224, 3)            │
0 │
```

```
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d (Conv2D)                      │ (None, 224, 224, 32)        │
896 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ batch_normalization                  │ (None, 224, 224, 32)        │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 112, 112, 32)        │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_1 (Conv2D)                    │ (None, 112, 112, 64)        │
18,496 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ batch_normalization_1                │ (None, 112, 112, 64)        │
256 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 56, 56, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_2 (Conv2D)                    │ (None, 56, 56, 128)         │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ batch_normalization_2                │ (None, 56, 56, 128)         │
512 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 28, 28, 128)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_3 (Conv2D)                    │ (None, 28, 28, 256)         │
295,168 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ batch_normalization_3                │ (None, 28, 28, 256)         │
1,024 │
```

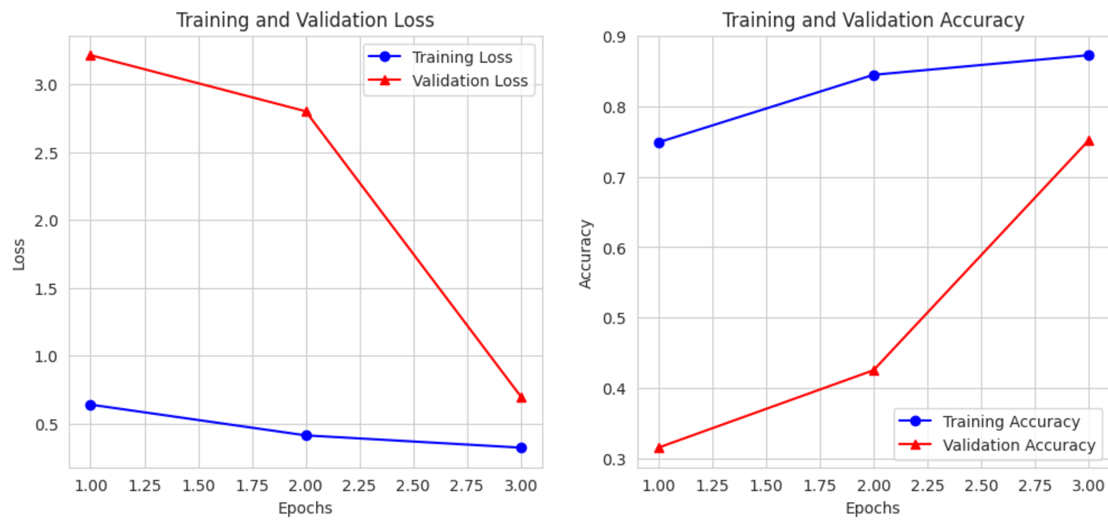
```
│ (BatchNormalization)                   │                              │
│
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ max_pooling2d_3 (MaxPooling2D)         │ (None, 14, 14, 256)          │
0 │
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ conv2d_4 (Conv2D)                      │ (None, 14, 14, 512)          │
1,180,160 │
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ batch_normalization_4                  │ (None, 14, 14, 512)          │
2,048 │
│ (BatchNormalization)                   │                              │
│
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ global_average_pooling2d               │ (None, 512)                  │
0 │
│ (GlobalAveragePooling2D)               │                              │
│
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ dense (Dense)                          │ (None, 512)                  │
262,656 │
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ batch_normalization_5                  │ (None, 512)                  │
2,048 │
│ (BatchNormalization)                   │                              │
│
├────────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ dense_1 (Dense)                        │ (None, 4)                    │
2,052 │
└────────────────────────────────────────┴──────────────────────────────┴───────
──────────┘

 Total params: 1,839,300 (7.02 MB)

 Trainable params: 1,836,292 (7.00 MB)

 Non-trainable params: 3,008 (11.75 KB)


--- Training with Adam for Comparison ---
Epoch 1/3
301/301 ━━━━━━━━━━━━━━━━━━━━━39s 86ms/step - accuracy: 0.6743 - loss: 0.8383
- val_accuracy: 0.3150 - val_loss: 3.2158
Epoch 2/3
```

```
301/301 ━━━━━━━━━━━━━━━━━━━━15s 48ms/step - accuracy: 0.8521 - loss: 0.3948
- val_accuracy: 0.4250 - val_loss: 2.8024
Epoch 3/3
301/301 ━━━━━━━━━━━━━━━━━━━━14s 46ms/step - accuracy: 0.8718 - loss: 0.3288
- val_accuracy: 0.7517 - val_loss: 0.6994

Model: "functional_1"

 ┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
━━━━━━━━━━┓
┃ Layer (type)                        ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
━━━━━━━━━━┩
│ input_layer_1 (InputLayer)          │ (None, 224, 224, 3)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_5 (Conv2D)                   │ (None, 224, 224, 32)        │
896 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ batch_normalization_6               │ (None, 224, 224, 32)        │
128 │
│ (BatchNormalization)                │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ max_pooling2d_4 (MaxPooling2D)      │ (None, 112, 112, 32)        │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_6 (Conv2D)                   │ (None, 112, 112, 64)        │
18,496 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ batch_normalization_7               │ (None, 112, 112, 64)        │
256 │
│ (BatchNormalization)                │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ max_pooling2d_5 (MaxPooling2D)      │ (None, 56, 56, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_7 (Conv2D)                   │ (None, 56, 56, 128)         │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
```

```
│ batch_normalization_8                 │ (None, 56, 56, 128)          │
512 │
│ (BatchNormalization)                  │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ max_pooling2d_6 (MaxPooling2D)        │ (None, 28, 28, 128)          │
0 │
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ conv2d_8 (Conv2D)                     │ (None, 28, 28, 256)          │
295,168 │
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ batch_normalization_9                 │ (None, 28, 28, 256)          │
1,024 │
│ (BatchNormalization)                  │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ max_pooling2d_7 (MaxPooling2D)        │ (None, 14, 14, 256)          │
0 │
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ conv2d_9 (Conv2D)                     │ (None, 14, 14, 512)          │
1,180,160 │
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ batch_normalization_10                │ (None, 14, 14, 512)          │
2,048 │
│ (BatchNormalization)                  │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ global_average_pooling2d_1            │ (None, 512)                  │
0 │
│ (GlobalAveragePooling2D)              │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ dense_2 (Dense)                       │ (None, 512)                  │
262,656 │
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ batch_normalization_11                │ (None, 512)                  │
2,048 │
│ (BatchNormalization)                  │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
```

```
│ dense_3 (Dense)                        │ (None, 4)                    │
2,052 │
└────────────────────────────────────────┴──────────────────────────────┴────────
──────────┘

 Total params: 5,511,886 (21.03 MB)

 Trainable params: 1,836,292 (7.00 MB)

 Non-trainable params: 3,008 (11.75 KB)

 Optimizer params: 3,672,586 (14.01 MB)
```

```
def plot_training_history(history):
    epochs = range(1, len(history.history['loss']) + 1)

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

    # Plot training & validation loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['loss'], 'bo-', label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], 'r^-', label='Validation
Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['accuracy'], 'bo-', label='Training
Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], 'r^-',
label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    plt.show()

plot_training_history(history_adam)
```




```
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix

test_gen_new.reset()

y_true = test_gen_new.classes

y_pred_probs = model_adam.predict(test_gen_new, steps=len(test_gen_new))

y_pred = np.argmax(y_pred_probs, axis=1)

print("Classification Report:")
print(classification_report(y_true, y_pred))

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=range(NUM_CLASSES), yticklabels=range(NUM_CLASSES))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
38/38 ━━━━━━━━━━━━━━━━━━━━7s 173ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.70      0.77       150
           1       0.52      0.97      0.68       150
           2       0.98      0.68      0.80       151
           3       0.98      0.64      0.77       150
```

	precision	recall	f1-score	support
accuracy			0.75	601
macro avg	0.84	0.75	0.76	601
weighted avg	0.84	0.75	0.76	601