Soil Image Classification using DeepBrainNet Architecture


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

base_path_train = "/kaggle/input/soil-dataset/Soil Train/Soil Train/"
base_path_test = "/kaggle/input/soil-dataset/Soil Test/Soil Test/"

def create_dataframe(base_path):
    data = {'image_path': [], 'label': []}
    for folder in os.listdir(base_path):
        folder_path = os.path.join(base_path, folder)
        if os.path.isdir(folder_path):
            for img in os.listdir(folder_path):
                if img.endswith(('.jpg', '.jpeg', '.png')):
                    data['image_path'].append(os.path.join(folder_path, img))
                    data['label'].append(folder)
    return pd.DataFrame(data)

train_df = create_dataframe(base_path_train)
```

```
test_df = create_dataframe(base_path_test)
df = pd.concat([train_df, test_df], ignore_index=True)
```

```
df.head()
```

```
                                          image_path          label
0  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
1  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
2  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
3  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
4  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
```

```
df.tail()
```

```
                                           image_path        label
363  /kaggle/input/soil-dataset/Soil Test/Soil Test...  Cinder Soil
364  /kaggle/input/soil-dataset/Soil Test/Soil Test...  Cinder Soil
365  /kaggle/input/soil-dataset/Soil Test/Soil Test...  Cinder Soil
366  /kaggle/input/soil-dataset/Soil Test/Soil Test...  Cinder Soil
367  /kaggle/input/soil-dataset/Soil Test/Soil Test...  Cinder Soil
```

```
df.shape
```

```
(368, 2)
```

```
df.columns
```

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

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

```
0
```

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

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 368 entries, 0 to 367
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  368 non-null    object
 1   label       368 non-null    object
dtypes: object(2)
memory usage: 5.9+ KB
```

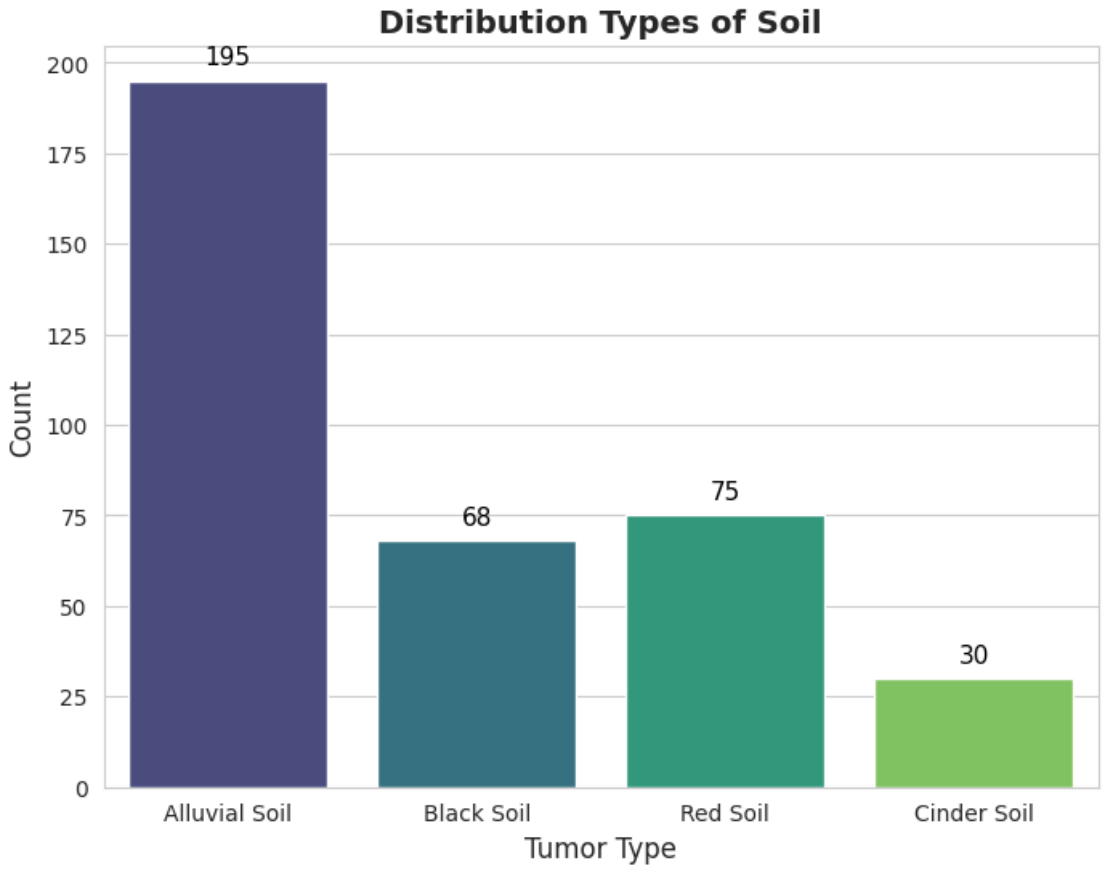
```
df['label'].unique()
```

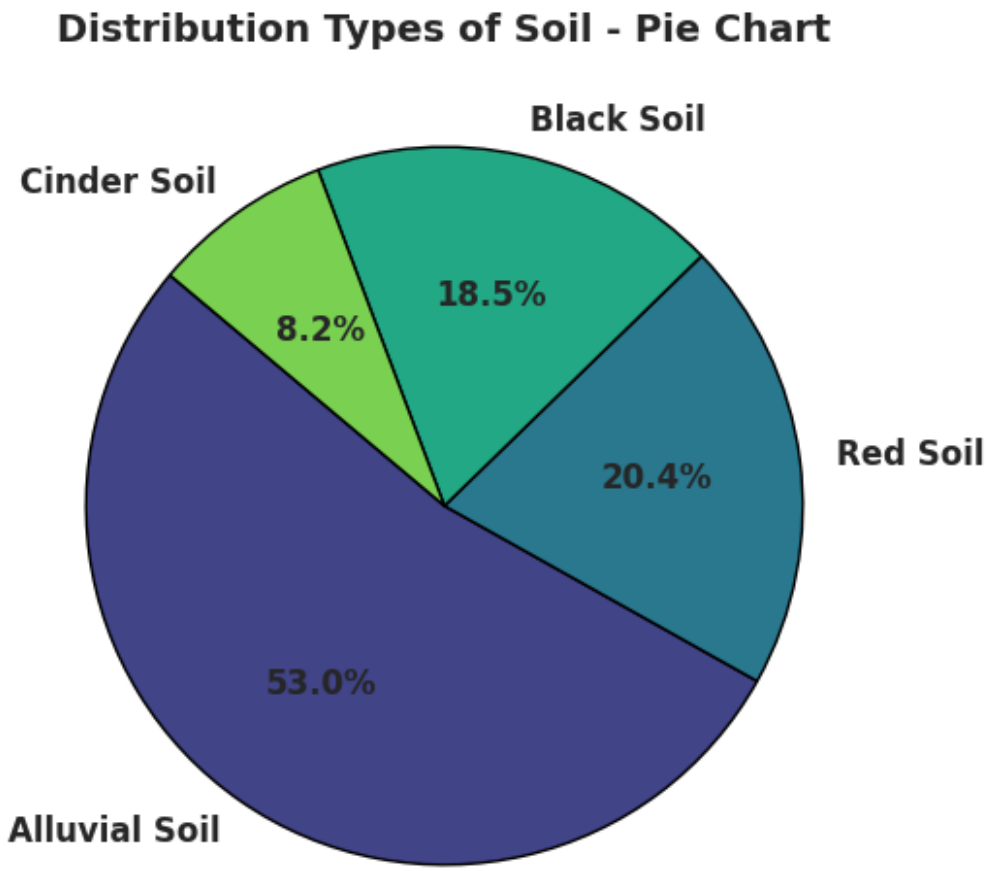
```
array(['Alluvial Soil', 'Black Soil', 'Red Soil', 'Cinder Soil'],
      dtype=object)
```

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

```
label
Alluvial Soil    195
Red Soil          75
Black Soil        68
Cinder Soil       30
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 Types of Soil", 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 Types of Soil - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution Types of Soil
Count
Tumor Type
0
25
50
75
100
125
150
175
200
195
68
75
30
Alluvial Soil
Black Soil
Red Soil
Cinder Soil




```
from PIL import Image

labels = combined_df['label'].unique()
fig, axes = plt.subplots(len(labels), 5, figsize=(15, 3*len(labels)))

for i, label in enumerate(labels):
    label_images = combined_df[combined_df['label'] ==
label]['image_path'].head(5).values
    for j, img_path in enumerate(label_images):
        img = Image.open(img_path)
        axes[i, j].imshow(img)
        axes[i, j].axis('off')
        if j == 0:
            axes[i, j].set_ylabel(label, rotation=0, labelpad=40,
fontsize=12)

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


```
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_balanced
```

	image_path	label
0	/kaggle/input/soil-dataset/Soil Train/Soil Tra...	Cinder Soil
1	/kaggle/input/soil-dataset/Soil Test/Soil Test...	Cinder Soil
2	/kaggle/input/soil-dataset/Soil Train/Soil Tra...	Red Soil
3	/kaggle/input/soil-dataset/Soil Train/Soil Tra...	Cinder Soil

```
4    /kaggle/input/soil-dataset/Soil Train/Soil Tra...       Red Soil
..                                                 ...            ...
775  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
776  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil
777  /kaggle/input/soil-dataset/Soil Train/Soil Tra...     Black Soil
778  /kaggle/input/soil-dataset/Soil Train/Soil Tra...       Red Soil
779  /kaggle/input/soil-dataset/Soil Train/Soil Tra...  Alluvial Soil

[780 rows x 2 columns]
```

```
df = df_balanced

import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.applications import InceptionV3, Xception
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,
Concatenate, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import seaborn as sns

le = LabelEncoder()
df['label'] = le.fit_transform(df['label'])

def preprocess_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (299, 299))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    hist_eq = cv2.equalizeHist(cv2.cvtColor(img, cv2.COLOR_RGB2GRAY))
    img = np.stack([img[:,:,0], img[:,:,1], hist_eq], axis=-1)
    return img / 255.0

def fuzzy_c_means_clustering(img, n_clusters=4):
    pixels = img.reshape(-1, 3)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
    _, labels, centers = cv2.kmeans(pixels.astype(np.float32), n_clusters,
None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
    return centers

images = np.array([preprocess_image(path) for path in df['image_path']])
features = np.array([fuzzy_c_means_clustering(img) for img in images])
labels = df['label'].values

train_idx, test_idx = train_test_split(df.index, test_size=0.2,
```

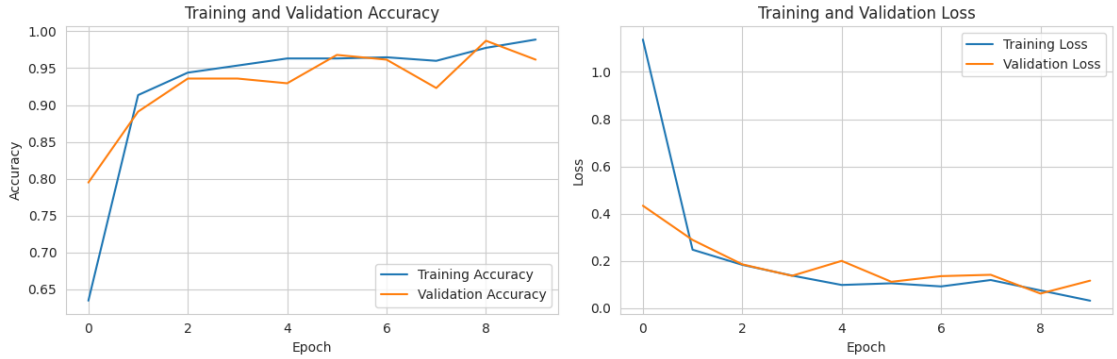
```
random_state=42)

input_shape = (299, 299, 3)
input_layer = Input(shape=input_shape)

inception_base = InceptionV3(weights='imagenet', include_top=False,
input_tensor=input_layer)
xception_base = Xception(weights='imagenet', include_top=False,
input_tensor=input_layer)

inception_out = GlobalAveragePooling2D()(inception_base.output)
xception_out = GlobalAveragePooling2D()(xception_base.output)
combined = Concatenate()([inception_out, xception_out])
dense = Dense(512, activation='relu')(combined)
output = Dense(4, activation='softmax')(dense)

model = Model(inputs=input_layer, outputs=output)

for layer in inception_base.layers:
    layer.trainable = False
for layer in xception_base.layers:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

train_images = np.array([preprocess_image(path) for path in df.loc[train_idx,
'image_path']])
test_images = np.array([preprocess_image(path) for path in df.loc[test_idx,
'image_path']])
y_train = df.loc[train_idx, 'label'].values
y_test = df.loc[test_idx, 'label'].values

history = model.fit(
    datagen.flow(train_images, y_train, batch_size=32),
    epochs=10,
    validation_data=(test_images, y_test)
)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()

y_pred = model.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)

cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[str(x) for x
in le.classes_], yticklabels=[str(x) for x in le.classes_])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred_classes, target_names=[str(x) for
x in le.classes_], zero_division=0))
```

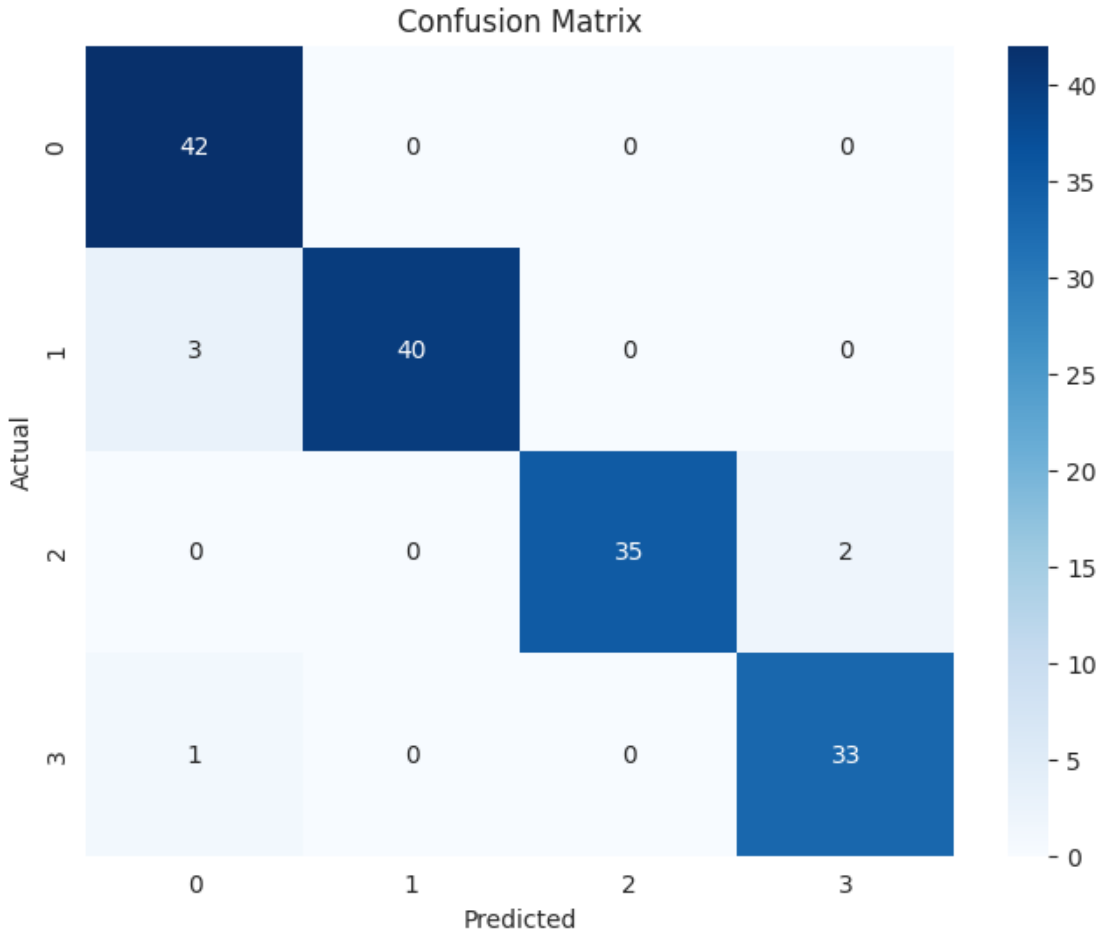
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_not
op.h5
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

/usr/local/lib/python3.11/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)`
in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

Epoch 1/10
```

```
2025-07-10 09:24:58.732633: E
external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm
eng3{k11=0} for conv (f32[32,256,74,74]{3,2,1,0}, u8[0]{0}) custom-
call(f32[32,256,74,74]{3,2,1,0}, f32[256,256,1,1]{3,2,1,0}),
window={size=1x1}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convForward",
backend_config={"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_
result_scale":1,"leakyrelu_alpha":0,"side_input_scale":0},"force_earliest_sch
edule":false,"operation_queue_id":"0","wait_on_operation_queues":[]} is
taking a while...
2025-07-10 09:24:58.980797: E
external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation
took 1.248265849s
Trying algorithm eng3{k11=0} for conv (f32[32,256,74,74]{3,2,1,0}, u8[0]{0})
custom-call(f32[32,256,74,74]{3,2,1,0}, f32[256,256,1,1]{3,2,1,0}),
window={size=1x1}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convForward",
backend_config={"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_
result_scale":1,"leakyrelu_alpha":0,"side_input_scale":0},"force_earliest_sch
edule":false,"operation_queue_id":"0","wait_on_operation_queues":[]} is
taking a while...

20/20 ━━━━━━━━━━━━━━━━━━━━ 112s 3s/step - accuracy: 0.4556 - loss: 1.8864 -
val_accuracy: 0.7949 - val_loss: 0.4338
Epoch 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 688ms/step - accuracy: 0.9044 - loss: 0.2985 -
val_accuracy: 0.8910 - val_loss: 0.2890
Epoch 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 689ms/step - accuracy: 0.9423 - loss: 0.1803 -
val_accuracy: 0.9359 - val_loss: 0.1857
Epoch 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 716ms/step - accuracy: 0.9613 - loss: 0.1284 -
val_accuracy: 0.9359 - val_loss: 0.1370
Epoch 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 700ms/step - accuracy: 0.9662 - loss: 0.0942 -
val_accuracy: 0.9295 - val_loss: 0.2003
Epoch 6/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 709ms/step - accuracy: 0.9710 - loss: 0.1008 -
val_accuracy: 0.9679 - val_loss: 0.1119
Epoch 7/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 680ms/step - accuracy: 0.9597 - loss: 0.0908 -
val_accuracy: 0.9615 - val_loss: 0.1357
Epoch 8/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 669ms/step - accuracy: 0.9570 - loss: 0.1150 -
val_accuracy: 0.9231 - val_loss: 0.1414
Epoch 9/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 691ms/step - accuracy: 0.9774 - loss: 0.0817 -
val_accuracy: 0.9872 - val_loss: 0.0620
Epoch 10/10
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 697ms/step - accuracy: 0.9864 - loss: 0.0356 -
val_accuracy: 0.9615 - val_loss: 0.1167
```




```
5/5 ━━━━━━━━━━━━━━━━━━━━ 21s 3s/step
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        42
```

1	1.00	0.93	0.96	43
2	1.00	0.95	0.97	37
3	0.94	0.97	0.96	34
accuracy			0.96	156
macro avg	0.96	0.96	0.96	156
weighted avg	0.96	0.96	0.96	156