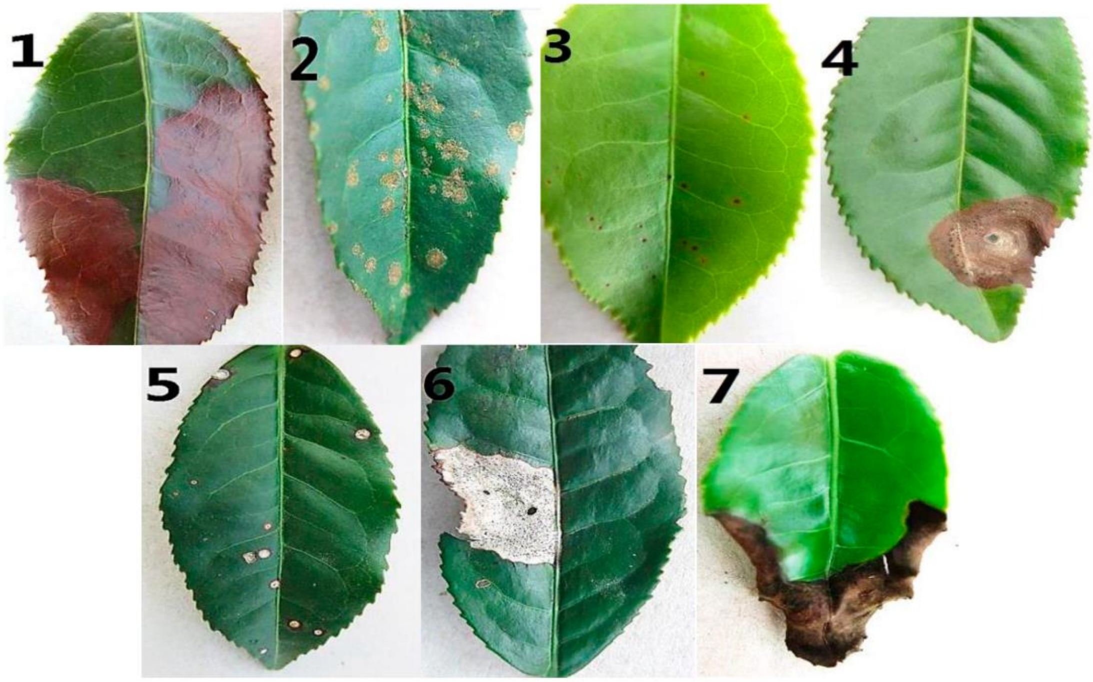
Tea Leaf Classification




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

base_path = "/kaggle/input/tealeafbd-tea-leaf-disease-detection/teaLeafBD/teaLeafBD"
categories = ["1. Tea algal leaf spot", "2. Brown Blight", "3. Gray Blight", "4. Helopeltis" , "5. Red spider", "6. Green mirid bug", "7. Healthy leaf"]

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)

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

df.head()
```

```
                                          image_path                  label
0  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  1. Tea algal leaf spot
1  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  1. Tea algal leaf spot
2  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  1. Tea algal leaf spot
3  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  1. Tea algal leaf spot
4  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  1. Tea algal leaf spot
```

```
df.tail()
```

```
                                             image_path            label
5271  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  7. Healthy leaf
5272  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  7. Healthy leaf
5273  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  7. Healthy leaf
5274  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  7. Healthy leaf
5275  /kaggle/input/tealeafbd-tea-leaf-disease-detec...  7. Healthy leaf
```

```
df.shape
```

```
(5276, 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: 5276 entries, 0 to 5275
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  5276 non-null   object
 1   label       5276 non-null   object
dtypes: object(2)
memory usage: 82.6+ KB
```

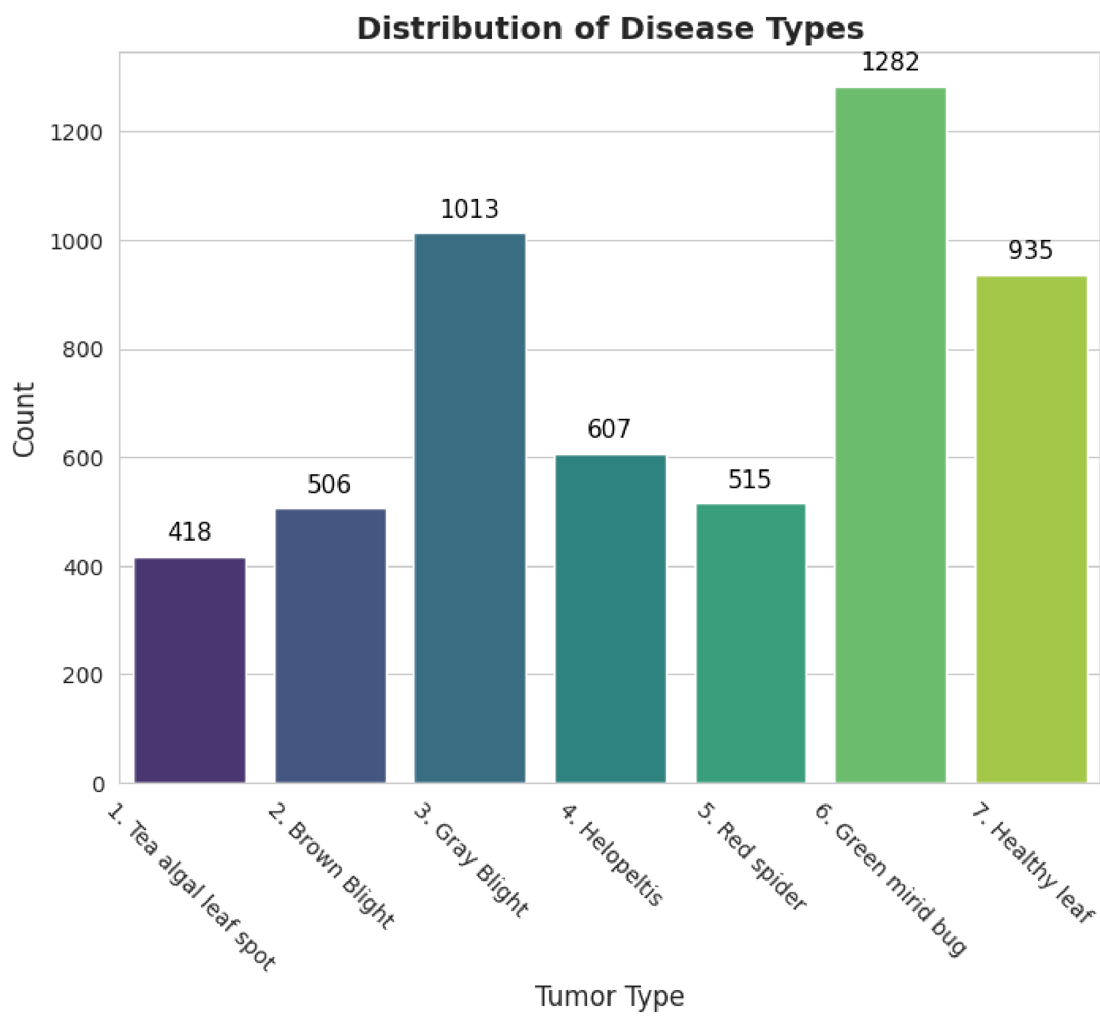
```
df['label'].unique()
```

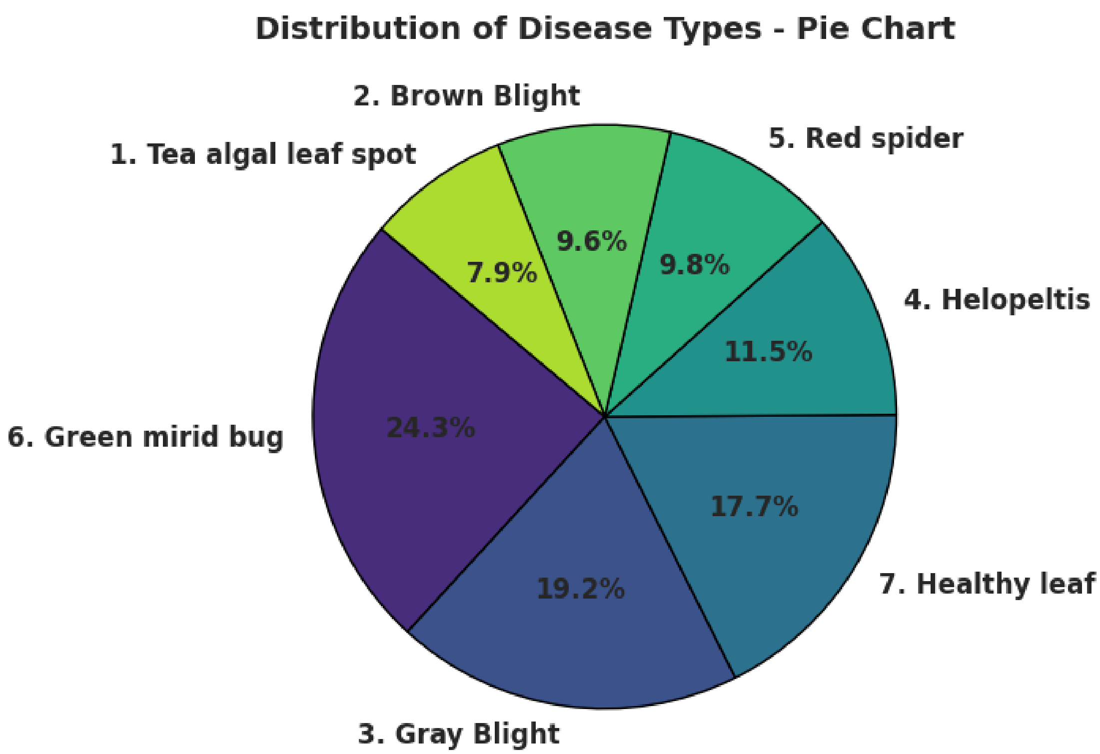
```
array(['1. Tea algal leaf spot', '2. Brown Blight', '3. Gray Blight',
       '4. Helopeltis', '5. Red spider', '6. Green mirid bug',
       '7. Healthy leaf'], dtype=object)
```

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

```
label
6. Green mirid bug        1282
3. Gray Blight            1013
7. Healthy leaf            935
4. Helopeltis              607
5. Red spider              515
2. Brown Blight            506
1. Tea algal leaf spot     418
Name: count, dtype: int64
```

```python
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 Disease 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.xticks(rotation=-45)
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 Disease Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of Disease Types
1282
1013
935
607
506
515
418
Count
1200
1000
800
600
400
200
0
1. Tea algal leaf spot
2. Brown Blight
3. Gray Blight
4. Helopeltis
5. Red spider
6. Green mirid bug
7. Healthy leaf
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.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['category_encoded'] = label_encoder.fit_transform(df['label'])

df = df[['image_path', 'category_encoded']]

from sklearn.utils import resample

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

dfs = []
for category in df['category_encoded'].unique():
    class_subset = df[df['category_encoded'] == 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['category_encoded'].value_counts()
```

```
category_encoded
3    1282
0    1282
1    1282
2    1282
4    1282
5    1282
6    1282
Name: count, dtype: int64
```

```
df_resampled = df_balanced
```

```
df_resampled['category_encoded'] =
df_resampled['category_encoded'].astype(str)
```

```
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')
```

```
2025-06-09 06:31:21.834484: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin cuFFT when
one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
E0000 00:00:1749450682.331013      35 cuda_dnn.cc:8310] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
E0000 00:00:1749450682.459893      35 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
```

```
check
```

```python
train_df_new, temp_df_new = train_test_split(
    df_resampled,
    train_size=0.8,
    shuffle=True,
    random_state=42,
    stratify=df_resampled['category_encoded']
)

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['category_encoded']
)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

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='category_encoded',
    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='category_encoded',
    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='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size
)
```

```
Found 7179 validated image filenames belonging to 7 classes.
Found 897 validated image filenames belonging to 7 classes.
Found 898 validated image filenames belonging to 7 classes.
```

```
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
```

```
from tensorflow.keras import layers, models

num_classes = 7

class ContinuousLayer(layers.Layer):
    def __init__(self, kernel_size=5, num_basis=10, output_channels=16,
**kwargs):
        super(ContinuousLayer, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.num_basis = num_basis
        self.output_channels = output_channels
        self.centers = self.add_weight(
            name='centers',
            shape=(num_basis, 2),
            initializer='random_normal',
            trainable=True
        )
        self.widths = self.add_weight(
            name='widths',
            shape=(num_basis,),
```

```
            initializer='ones',
            trainable=True,
            constraint=tf.keras.constraints.NonNeg()
        )
        self.kernel_weights = self.add_weight(
            name='kernel_weights',
            shape=(kernel_size, kernel_size, channels, output_channels),
            initializer='glorot_normal',
            trainable=True
        )

    def call(self, inputs):
        height, width = img_size
        x = tf.range(0, height, 1.0)
        y = tf.range(0, width, 1.0)
        x_grid, y_grid = tf.meshgrid(x, y)
        grid = tf.stack([x_grid, y_grid], axis=-1)

        basis = []
        for i in range(self.num_basis):
            center = self.centers[i]
            width = self.widths[i]
            dist = tf.reduce_sum(((grid - center) / width) ** 2, axis=-1)
            basis_i = tf.exp(-dist)
            basis.append(basis_i)
        basis = tf.stack(basis, axis=-1)

        basis_weights = tf.reduce_mean(basis, axis=[0, 1])
        basis_weights = tf.nn.softmax(basis_weights)
        basis_weights = basis_weights[:, tf.newaxis, tf.newaxis, tf.newaxis,
tf.newaxis]

        modulated_kernel = self.kernel_weights * tf.reduce_sum(basis_weights,
axis=0)

        output = tf.nn.conv2d(
            inputs,
            modulated_kernel,
            strides=[1, 1, 1, 1],
            padding='SAME'
        )

        return output

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[1], input_shape[2],
self.output_channels)

    def smoothness_penalty(self):
```

```
        grad_x = tf.reduce_mean(tf.square(self.kernel_weights[1:, :, :, :] -
self.kernel_weights[:-1, :, :, :]))
        grad_y = tf.reduce_mean(tf.square(self.kernel_weights[:, 1:, :, :] -
self.kernel_weights[:, :-1, :, :]))
        return grad_x + grad_y

class VariationalLoss(tf.keras.losses.Loss):
    def __init__(self, model, lambda1=0.01, lambda2=1.0):
        super(VariationalLoss, self).__init__()
        self.model = model
        self.lambda1 = lambda1
        self.lambda2 = lambda2
        self.sce = tf.keras.losses.SparseCategoricalCrossentropy()  # Changed
to SparseCategoricalCrossentropy

    def call(self, y_true, y_pred):
        smoothness_penalty = 0
        for layer in self.model.layers:
            if isinstance(layer, ContinuousLayer):
                smoothness_penalty += layer.smoothness_penalty()
        prediction_loss = self.sce(y_true, y_pred)
        return self.lambda2 * prediction_loss + self.lambda1 *
smoothness_penalty

def build_continuous_model():
    inputs = layers.Input(shape=img_shape)
    x = ContinuousLayer(kernel_size=5, num_basis=10,
output_channels=16)(inputs)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)  # Changed
to 7 units with softmax
    model = models.Model(inputs, outputs)
    return model

model = build_continuous_model()

model.compile(
    optimizer='adam',
    loss=VariationalLoss(model=model, lambda1=0.01, lambda2=1.0),
    metrics=['accuracy']
)

history = model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
```

```
    epochs=3,
    verbose=1
)
```

```
Epoch 1/3

WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1749451063.329868     141 service.cc:148] XLA service
0x7da2d8025740 initialized for platform CUDA (this does not guarantee that
XLA will be used). Devices:
I0000 00:00:1749451063.331201     141 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1749451063.331220     141 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1749451063.860056     141 cuda_dnn.cc:529] Loaded cuDNN version
90300

  1/449 ━━━━━━━━━━━━━━━━━━━━ 1:15:43 10s/step - accuracy: 0.1875 - loss:
1.9956

I0000 00:00:1749451069.707228     141 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

449/449 ━━━━━━━━━━━━━━━━━━━━ 138s 285ms/step - accuracy: 0.2923 - loss:
2.3807 - val_accuracy: 0.5953 - val_loss: 1.1260
Epoch 2/3
449/449 ━━━━━━━━━━━━━━━━━━━━ 83s 185ms/step - accuracy: 0.6487 - loss: 1.0014
- val_accuracy: 0.7915 - val_loss: 0.6917
Epoch 3/3
449/449 ━━━━━━━━━━━━━━━━━━━━ 84s 187ms/step - accuracy: 0.8393 - loss: 0.5007
- val_accuracy: 0.8506 - val_loss: 0.4932
```

```
model.summary()
```

```
Model: "functional"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
━━━━━━━━━━┩
│ input_layer_2 (InputLayer)            │ (None, 224, 224, 3)          │
0 │
├───────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ continuous_layer_2 (ContinuousLayer) │ (None, 224, 224, 16)          │
1,230 │
├───────────────────────────────────────┼──────────────────────────────┼───────
──────────┤
│ activation_2 (Activation)             │ (None, 224, 224, 16)         │
```

```
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 112, 112, 16)        │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ flatten_2 (Flatten)                  │ (None, 200704)              │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ dense_3 (Dense)                      │ (None, 128)                 │
25,690,240 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ dropout_2 (Dropout)                  │ (None, 128)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ dense_4 (Dense)                      │ (None, 7)                   │
903 │
└──────────────────────────────────────┴─────────────────────────────┴───────
──────────┘

 Total params: 77,077,121 (294.03 MB)

 Trainable params: 25,692,373 (98.01 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 51,384,748 (196.02 MB)
```

```python
test_loss, test_accuracy = model.evaluate(test_gen_new)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 12s 212ms/step - accuracy: 0.8856 - loss: 0.4414
Test Loss: 0.4905, Test Accuracy: 0.8719
```

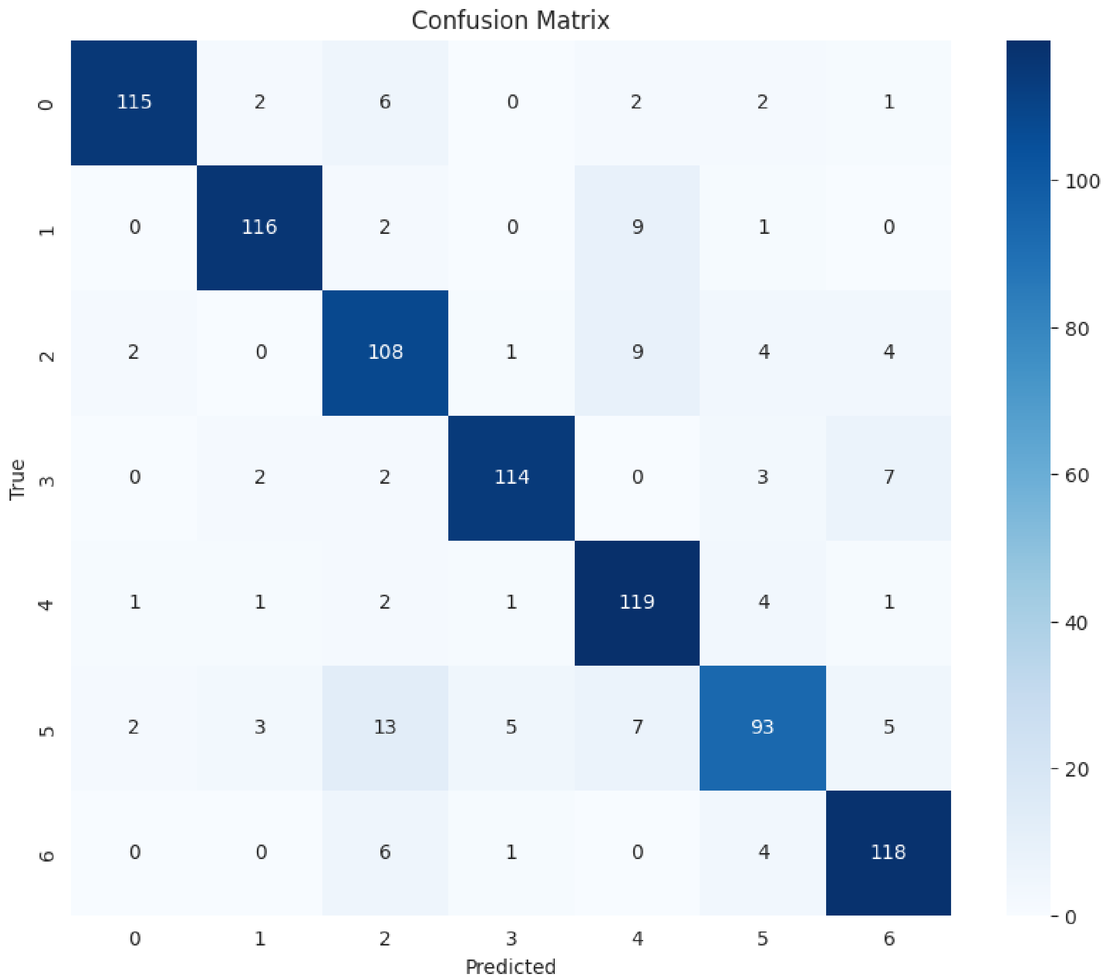
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

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


```
test_gen_new.reset()
y_pred = model.predict(test_gen_new)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_gen_new.classes
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 158ms/step
```

```
cm = confusion_matrix(y_true, y_pred_classes)

class_names = list(test_gen_new.class_indices.keys())

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
class ContinuousLayer(layers.Layer):
    def __init__(self, kernel_size=5, num_basis=10, output_channels=16,
reduction_ratio=4, **kwargs):
        super(ContinuousLayer, self).__init__(**kwargs)
        self.kernel_size = kernel_size
        self.num_basis = num_basis
        self.output_channels = output_channels
        self.reduction_ratio = reduction_ratio

        self.centers = self.add_weight(
            name='centers',
            shape=(num_basis, 2),
            initializer='random_normal',
            trainable=True
        )
        self.widths = self.add_weight(
            name='widths',
            shape=(num_basis,),
            initializer='ones',
            trainable=True,
```

```
            constraint=tf.keras.constraints.NonNeg()
        )

        self.kernel_weights = self.add_weight(
            name='kernel_weights',
            shape=(kernel_size, kernel_size, channels, output_channels),
            initializer='glorot_normal',
            trainable=True
        )

        self.attention_pooling = layers.GlobalAveragePooling2D()
        self.attention_dense1 = layers.Dense(
            max(num_basis // self.reduction_ratio, 1),
            activation='relu',
            name='attention_dense1'
        )

        self.attention_dense2 = layers.Dense(
            num_basis,
            activation='sigmoid',
            name='attention_dense2'
        )

    def call(self, inputs):
        height, width = img_size

        x_coords = tf.range(0, height, 1.0)
        y_coords = tf.range(0, width, 1.0)
        x_grid, y_grid = tf.meshgrid(x_coords, y_coords)
        grid = tf.stack([x_grid, y_grid], axis=-1)

        centers_reshaped = self.centers[tf.newaxis, tf.newaxis, :, :]
        widths_reshaped = self.widths[tf.newaxis, tf.newaxis, :, tf.newaxis]
        safe_widths = tf.maximum(widths_reshaped, tf.keras.backend.epsilon())

        diff = (grid[:, :, tf.newaxis, :] - centers_reshaped) / safe_widths
        dist_squared = tf.reduce_sum(diff ** 2, axis=-1)
        basis = tf.exp(-dist_squared)

        squeeze = self.attention_pooling(inputs)

        excitation = self.attention_dense1(squeeze)
        attention_weights = self.attention_dense2(excitation)

        mean_basis_activation = tf.reduce_mean(basis, axis=[0, 1]) #
(num_basis,)

        dynamic_basis_modulation = attention_weights *
mean_basis_activation[tf.newaxis, :] # (batch_size, num_basis)
```

```
        scaling_factor_per_batch = tf.reduce_sum(dynamic_basis_modulation,
axis=-1, keepdims=True) # (batch_size, 1)
        global_features = self.attention_pooling(inputs) # (batch_size,
input_channels)

        predicted_basis_weights_per_batch =
self.attention_dense1(global_features)
        predicted_basis_weights_per_batch =
self.attention_dense2(predicted_basis_weights_per_batch) # (batch_size,
num_basis)

        attended_basis_weights =
tf.reduce_mean(predicted_basis_weights_per_batch, axis=0) # (num_basis,)

        attended_basis_weights = tf.nn.softmax(attended_basis_weights)

        scaling_factor = tf.reduce_sum(attended_basis_weights)

        modulated_kernel = self.kernel_weights * scaling_factor

        output = tf.nn.conv2d(
            inputs,
            modulated_kernel,
            strides=[1, 1, 1, 1],
            padding='SAME'
        )

        return output

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[1], input_shape[2],
self.output_channels)

    def smoothness_penalty(self):
        grad_x = tf.reduce_mean(tf.square(self.kernel_weights[1:, :, :, :] -
self.kernel_weights[:-1, :, :, :]))
        grad_y = tf.reduce_mean(tf.square(self.kernel_weights[:, 1:, :, :] -
self.kernel_weights[:, :-1, :, :]))
        return grad_x + grad_y

class VariationalLoss(tf.keras.losses.Loss):
    def __init__(self, model, lambda1=0.01, lambda2=1.0):
        super(VariationalLoss, self).__init__()
        self.model = model
        self.lambda1 = lambda1
        self.lambda2 = lambda2
        self.sce = tf.keras.losses.SparseCategoricalCrossentropy()
```

```
    def call(self, y_true, y_pred):
        smoothness_penalty = 0
        for layer in self.model.layers:
            if isinstance(layer, ContinuousLayer):
                smoothness_penalty += layer.smoothness_penalty()
        prediction_loss = self.sce(y_true, y_pred)
        return self.lambda2 * prediction_loss + self.lambda1 *
smoothness_penalty

def build_continuous_model_with_attention():
    inputs = layers.Input(shape=img_shape)
    x = ContinuousLayer(kernel_size=5, num_basis=10, output_channels=16,
reduction_ratio=4)(inputs)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    model = models.Model(inputs, outputs)
    return model

model.summary()
```

```
Model: "functional_4"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━
━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                  ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━
━━━━━━━━━━┩
│ input_layer_6 (InputLayer)           │ (None, 224, 224, 3)           │
0 │
├───────────────────────────────────────┼──────────────────────────────┼────────
──────────┤
│ continuous_layer_8 (ContinuousLayer) │ (None, 224, 224, 16)          │
1,268 │
├───────────────────────────────────────┼──────────────────────────────┼────────
──────────┤
│ activation_8 (Activation)            │ (None, 224, 224, 16)          │
0 │
├───────────────────────────────────────┼──────────────────────────────┼────────
──────────┤
│ max_pooling2d_8 (MaxPooling2D)       │ (None, 112, 112, 16)          │
0 │
├───────────────────────────────────────┼──────────────────────────────┼────────
──────────┤
│ flatten_6 (Flatten)                  │ (None, 200704)                │
0 │
```

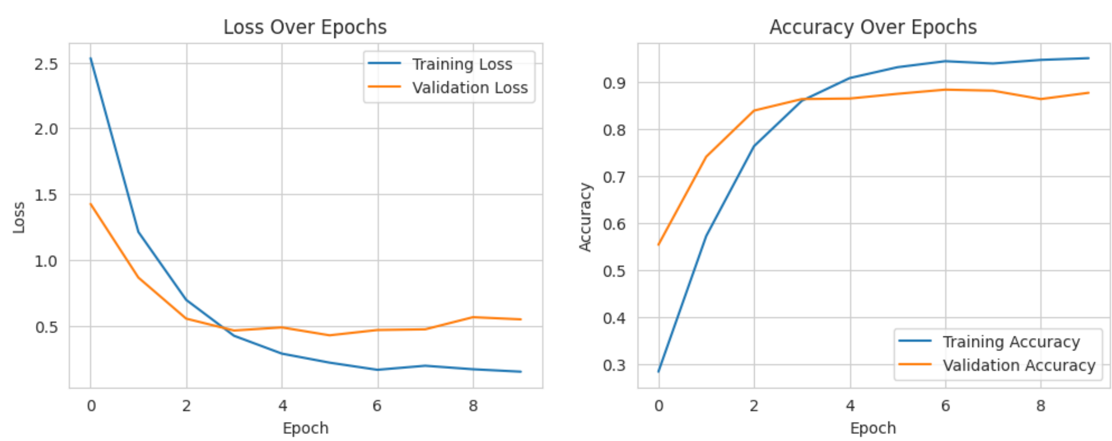
```
├──────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ dense_11 (Dense)                     │ (None, 128)                  │
25,690,240 │
├──────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ dropout_6 (Dropout)                  │ (None, 128)                  │
0 │
├──────────────────────────────────────┼──────────────────────────────┼───────
───────────┤
│ dense_12 (Dense)                     │ (None, 7)                    │
903 │
└──────────────────────────────────────┴──────────────────────────────┴───────
───────────┘

 Total params: 77,077,235 (294.03 MB)

 Trainable params: 25,692,411 (98.01 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 51,384,824 (196.02 MB)
```

```
model = build_continuous_model()

model.compile(
    optimizer='adam',
    loss=VariationalLoss(model=model, lambda1=0.01, lambda2=1.0),
    metrics=['accuracy']
)

history = model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
    epochs=10,
    verbose=1
)
```

```
Epoch 1/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 91s 193ms/step - accuracy: 0.2086 - loss: 5.5501
- val_accuracy: 0.5530 - val_loss: 1.4257
Epoch 2/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 83s 185ms/step - accuracy: 0.5187 - loss: 1.3297
- val_accuracy: 0.7402 - val_loss: 0.8669
Epoch 3/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 83s 186ms/step - accuracy: 0.7466 - loss: 0.7435
- val_accuracy: 0.8384 - val_loss: 0.5552
Epoch 4/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 84s 187ms/step - accuracy: 0.8535 - loss: 0.4453
- val_accuracy: 0.8629 - val_loss: 0.4652
Epoch 5/10
```

```
449/449 ━━━━━━━━━━━━━━━━━━━━ 83s 186ms/step - accuracy: 0.9069 - loss: 0.2940
- val_accuracy: 0.8640 - val_loss: 0.4886
Epoch 6/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 80s 179ms/step - accuracy: 0.9330 - loss: 0.2144
- val_accuracy: 0.8740 - val_loss: 0.4286
Epoch 7/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 84s 187ms/step - accuracy: 0.9487 - loss: 0.1594
- val_accuracy: 0.8829 - val_loss: 0.4688
Epoch 8/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 81s 181ms/step - accuracy: 0.9430 - loss: 0.1879
- val_accuracy: 0.8807 - val_loss: 0.4741
Epoch 9/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 83s 186ms/step - accuracy: 0.9462 - loss: 0.1729
- val_accuracy: 0.8629 - val_loss: 0.5662
Epoch 10/10
449/449 ━━━━━━━━━━━━━━━━━━━━ 84s 187ms/step - accuracy: 0.9525 - loss: 0.1502
- val_accuracy: 0.8763 - val_loss: 0.5499
```

```python
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
test_gen_new.reset()
y_pred = model.predict(test_gen_new)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_gen_new.classes
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 166ms/step
```

```
cm = confusion_matrix(y_true, y_pred_classes)
```

```
class_names = list(test_gen_new.class_indices.keys())
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```