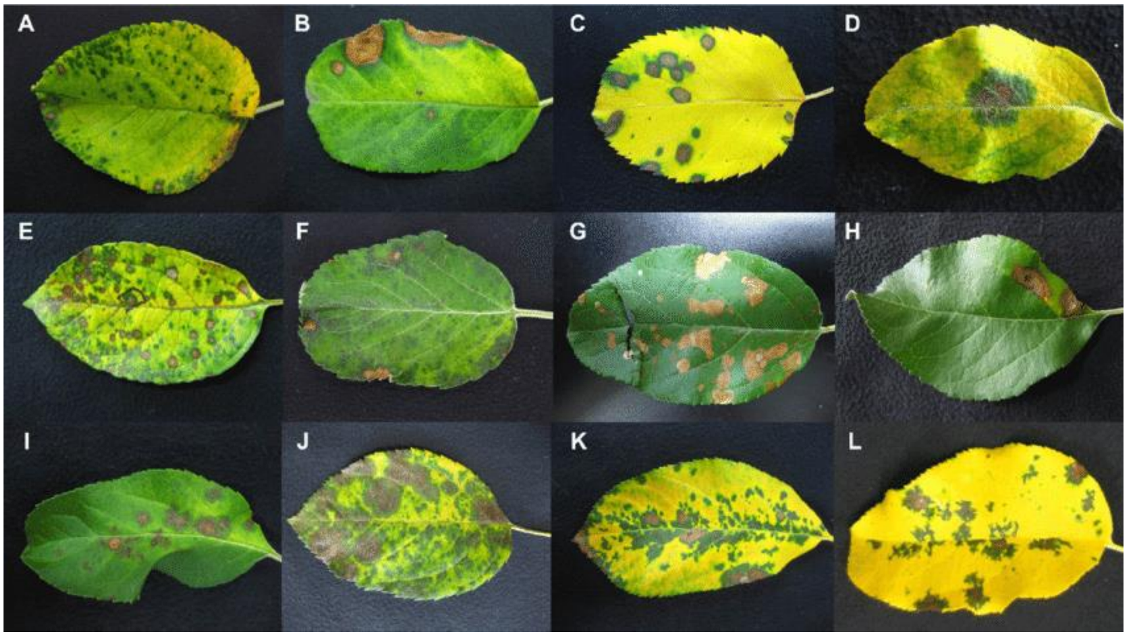
Apple Leaf Disease Detection



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

base_path = "/kaggle/input/leaf-disease-images/leaves/Apple"
categories = ["Apple_Black_rot", "Apple_healthy", "Apple_rust", "Apple_scab"]

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/leaf-disease-images/leaves/Apple...  Apple_Black_rot
1  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_Black_rot
```

```
2  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_Black_rot
3  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_Black_rot
4  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_Black_rot
```

```
df.tail()
```

```
                                             image_path       label
3159  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_scab
3160  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_scab
3161  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_scab
3162  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_scab
3163  /kaggle/input/leaf-disease-images/leaves/Apple...  Apple_scab
```

```
df.shape
```

```
(3164, 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: 3164 entries, 0 to 3163
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  3164 non-null   object
 1   label       3164 non-null   object
dtypes: object(2)
memory usage: 49.6+ KB
```

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

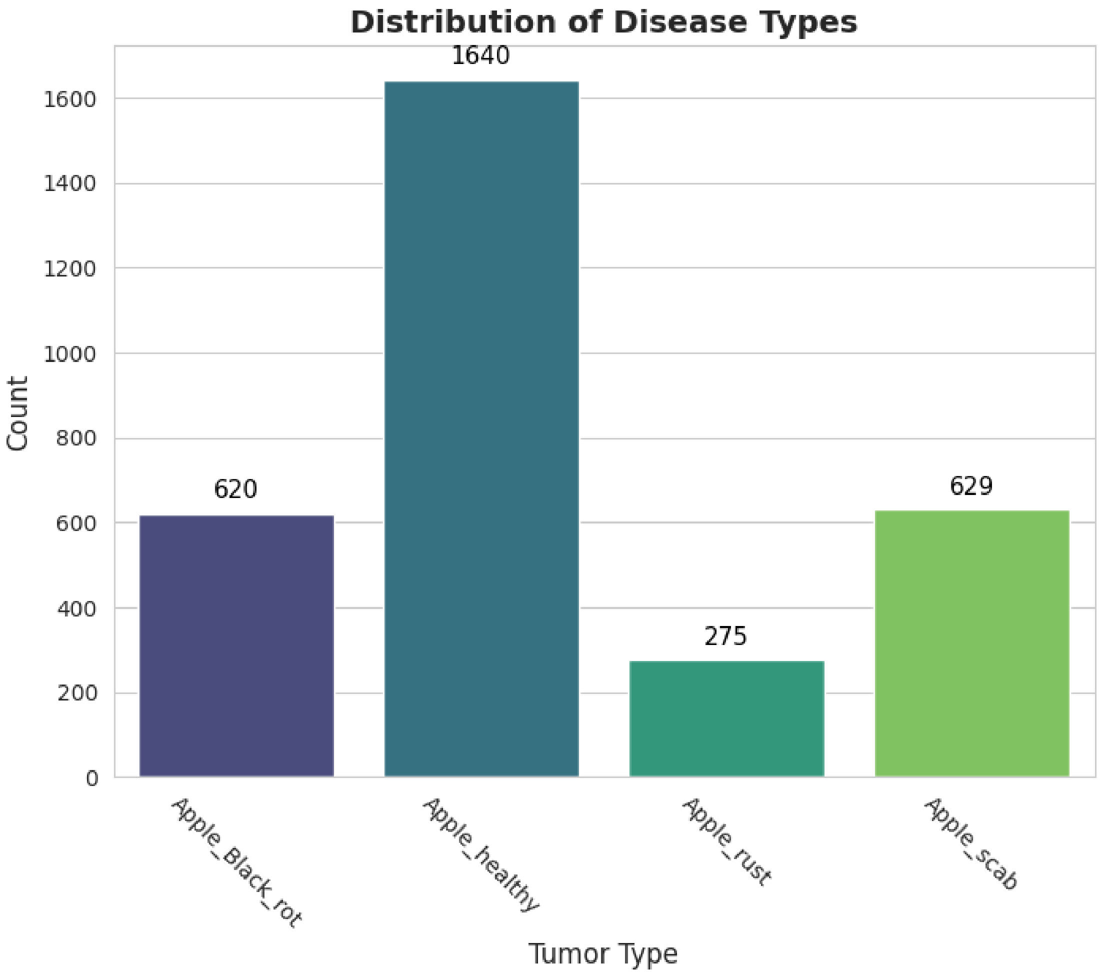
```
array(['Apple_Black_rot', 'Apple_healthy', 'Apple_rust', 'Apple_scab'],
      dtype=object)
```

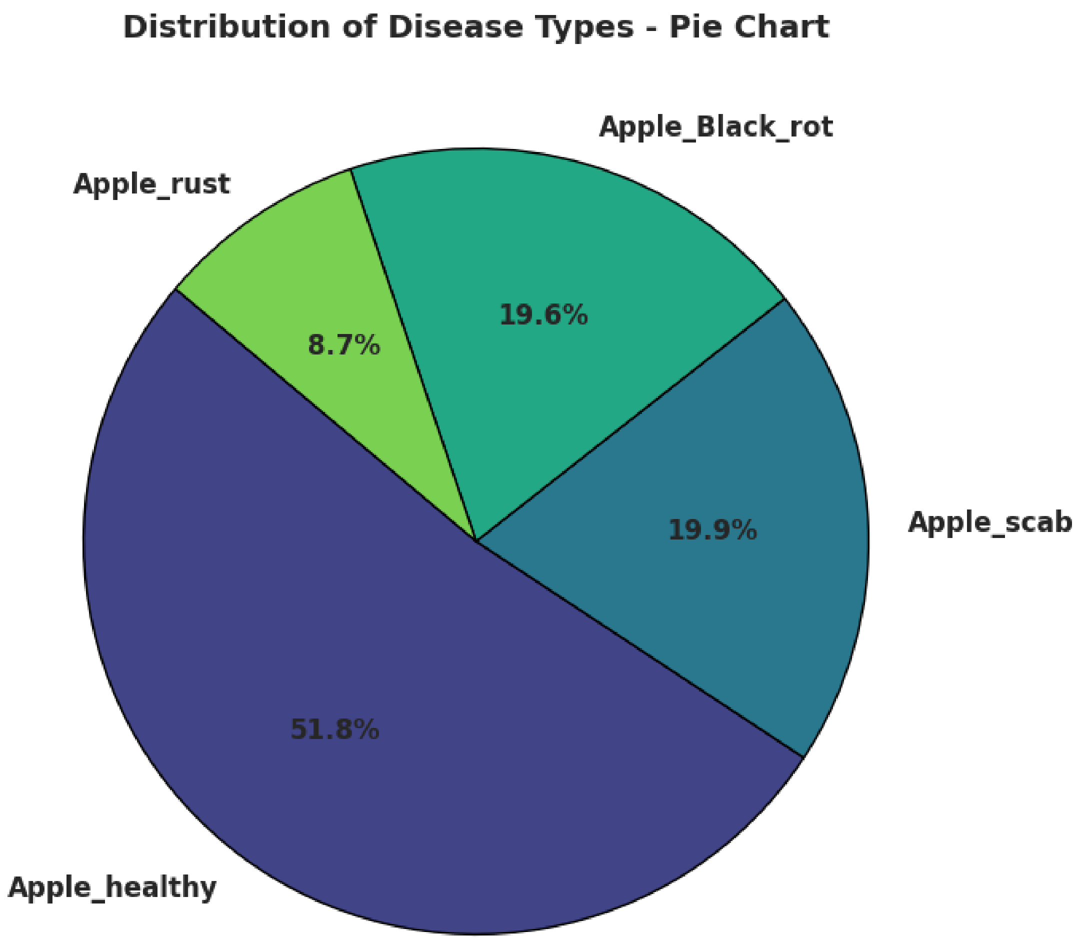
```
df['label'].value_counts()
```

```
label
Apple_healthy      1640
Apple_scab          629
```

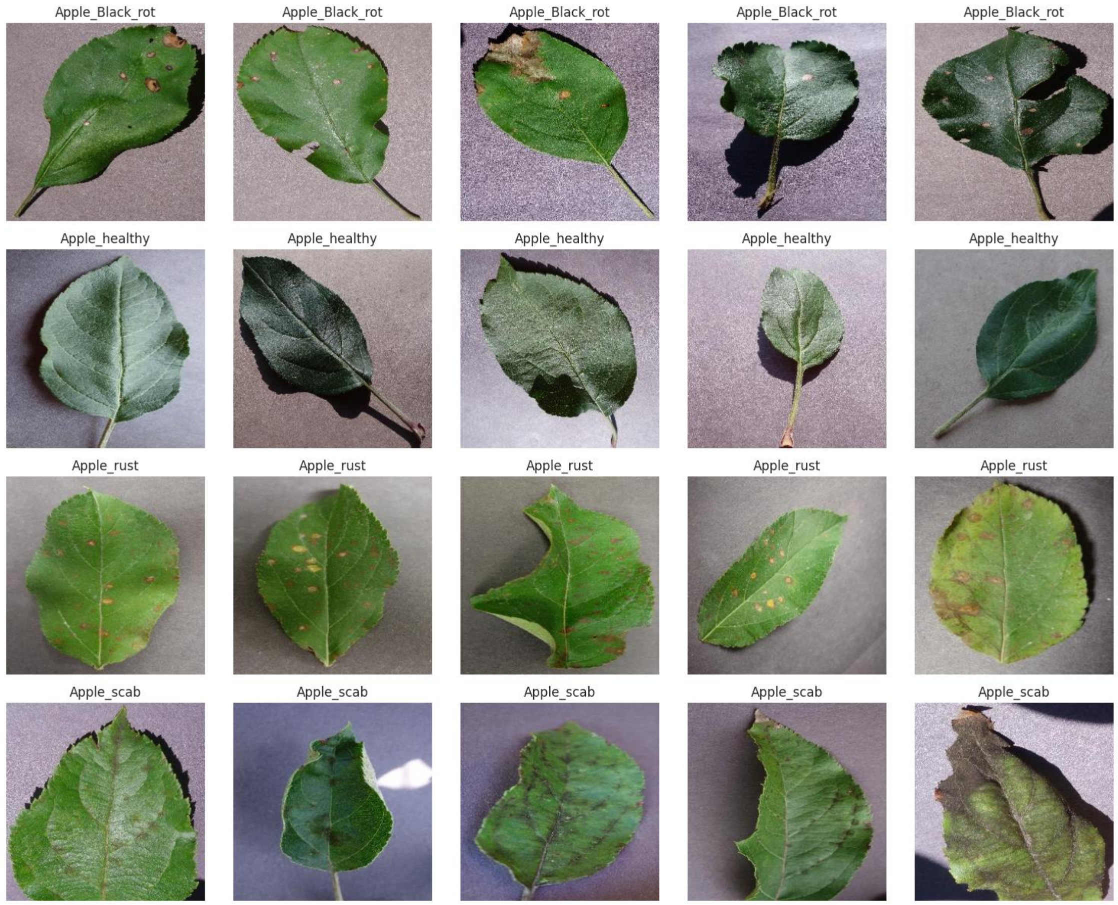
```
Apple_Black_rot     620
Apple_rust          275
Name: count, dtype: int64
```

```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=(20, 8))
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
Count
0
200
400
600
800
1000
1200
1400
1600
620
1640
275
629
Apple_Black_rot
Apple_healthy
Apple_rust
Apple_scab
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
1    1640
3    1640
2    1640
0    1640
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')
```

```
2025-06-10 12:32:53.447914: 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 00:00:1749558773.671901      35 cuda_dnn.cc:8310] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
E0000 00:00:1749558773.738947      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 5248 validated image filenames belonging to 4 classes.
Found 656 validated image filenames belonging to 4 classes.
Found 656 validated image filenames belonging to 4 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 = 4
```

```
import tensorflow as tf
from tensorflow.keras import layers, models


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, 32, output_channels),  # Updated
to 32 input 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()

    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 = layers.Conv2D(filters=32, kernel_size=3, padding='same',
activation='relu')(inputs)
    x = ContinuousLayer(kernel_size=5, num_basis=10, output_channels=16)(x)
    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 = 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

WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1749558926.317944      84 service.cc:148] XLA service
0x7e4790015780 initialized for platform CUDA (this does not guarantee that
XLA will be used). Devices:
I0000 00:00:1749558926.318897      84 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1749558926.318931      84 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1749558926.794642      84 cuda_dnn.cc:529] Loaded cuDNN version
90300

  3/328 ━━━━━━━━━━━━━━━━━━━━━20s 63ms/step - accuracy: 0.2812 - loss: 8.2690

I0000 00:00:1749558934.082973      84 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

328/328 ━━━━━━━━━━━━━━━━━━━━━33s 66ms/step - accuracy: 0.5818 - loss: 1.7665
- val_accuracy: 0.9131 - val_loss: 0.2266
Epoch 2/10
328/328 ━━━━━━━━━━━━━━━━━━━━━14s 43ms/step - accuracy: 0.9091 - loss: 0.2781
- val_accuracy: 0.9466 - val_loss: 0.1548
Epoch 3/10
328/328 ━━━━━━━━━━━━━━━━━━━━━13s 40ms/step - accuracy: 0.9532 - loss: 0.1428
- val_accuracy: 0.9558 - val_loss: 0.1355
Epoch 4/10
328/328 ━━━━━━━━━━━━━━━━━━━━━13s 41ms/step - accuracy: 0.9713 - loss: 0.0845
- val_accuracy: 0.9573 - val_loss: 0.1367
Epoch 5/10
328/328 ━━━━━━━━━━━━━━━━━━━━━13s 40ms/step - accuracy: 0.9734 - loss: 0.0738
- val_accuracy: 0.9573 - val_loss: 0.1654
Epoch 6/10
328/328 ━━━━━━━━━━━━━━━━━━━━━13s 40ms/step - accuracy: 0.9807 - loss: 0.0772
- val_accuracy: 0.9665 - val_loss: 0.1116
Epoch 7/10
328/328 ━━━━━━━━━━━━━━━━━━━━━14s 41ms/step - accuracy: 0.9889 - loss: 0.0395
- val_accuracy: 0.9619 - val_loss: 0.1281
Epoch 8/10
328/328 ━━━━━━━━━━━━━━━━━━━━━14s 41ms/step - accuracy: 0.9763 - loss: 0.0641
- val_accuracy: 0.9695 - val_loss: 0.0987
Epoch 9/10
```

```
328/328 ━━━━━━━━━━━━━━━━━━━━━14s 42ms/step - accuracy: 0.9717 - loss: 0.1013
- val_accuracy: 0.9726 - val_loss: 0.1221
Epoch 10/10
328/328 ━━━━━━━━━━━━━━━━━━━━━14s 41ms/step - accuracy: 0.9836 - loss: 0.0656
- val_accuracy: 0.9695 - val_loss: 0.1543
```

```
model.summary()
```

```
Model: "functional_1"
```

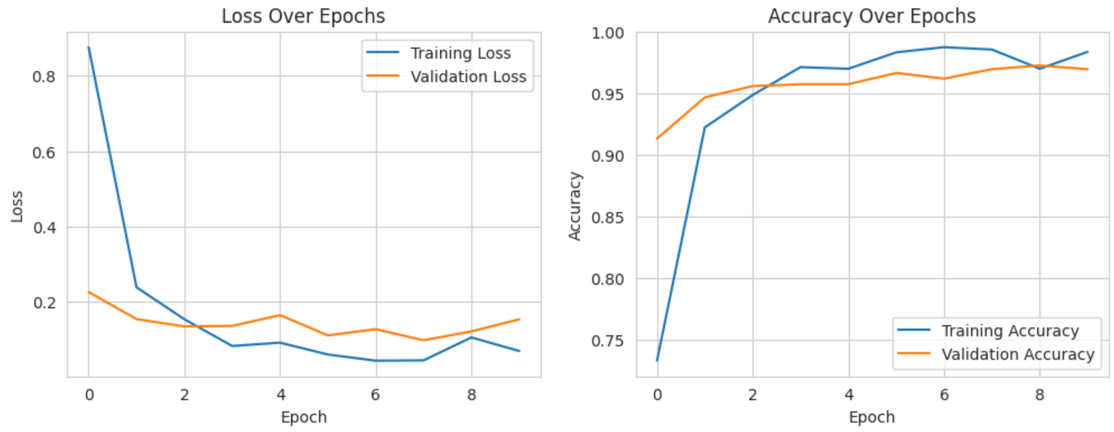
```
 ┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
━━━━━━━━━━┓
 ┃ Layer (type)                         ┃ Output Shape                ┃
Param #  ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
━━━━━━━━━━┩
│ input_layer_2 (InputLayer)          │ (None, 224, 224, 3)         │
0 │
├─────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ conv2d_2 (Conv2D)                   │ (None, 224, 224, 32)        │
896 │
├─────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ continuous_layer_2 (ContinuousLayer) │ (None, 224, 224, 16)        │
12,830 │
├─────────────────────────────────────┼─────────────────────────────┼───────
──────────┤
│ 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, 4)                   │
516 │
```

```
└──────────────────────────────────────┴──────────────────────────────┴────────
──────────┘

 Total params: 77,113,448 (294.16 MB)

 Trainable params: 25,704,482 (98.05 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 51,408,966 (196.11 MB)
```

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

```
41/41 ━━━━━━━━━━━━━━━━━━━━━━2s 43ms/step - accuracy: 0.9861 - loss: 0.0634
Test Loss: 0.0620, Test Accuracy: 0.9863
```

```
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)
```

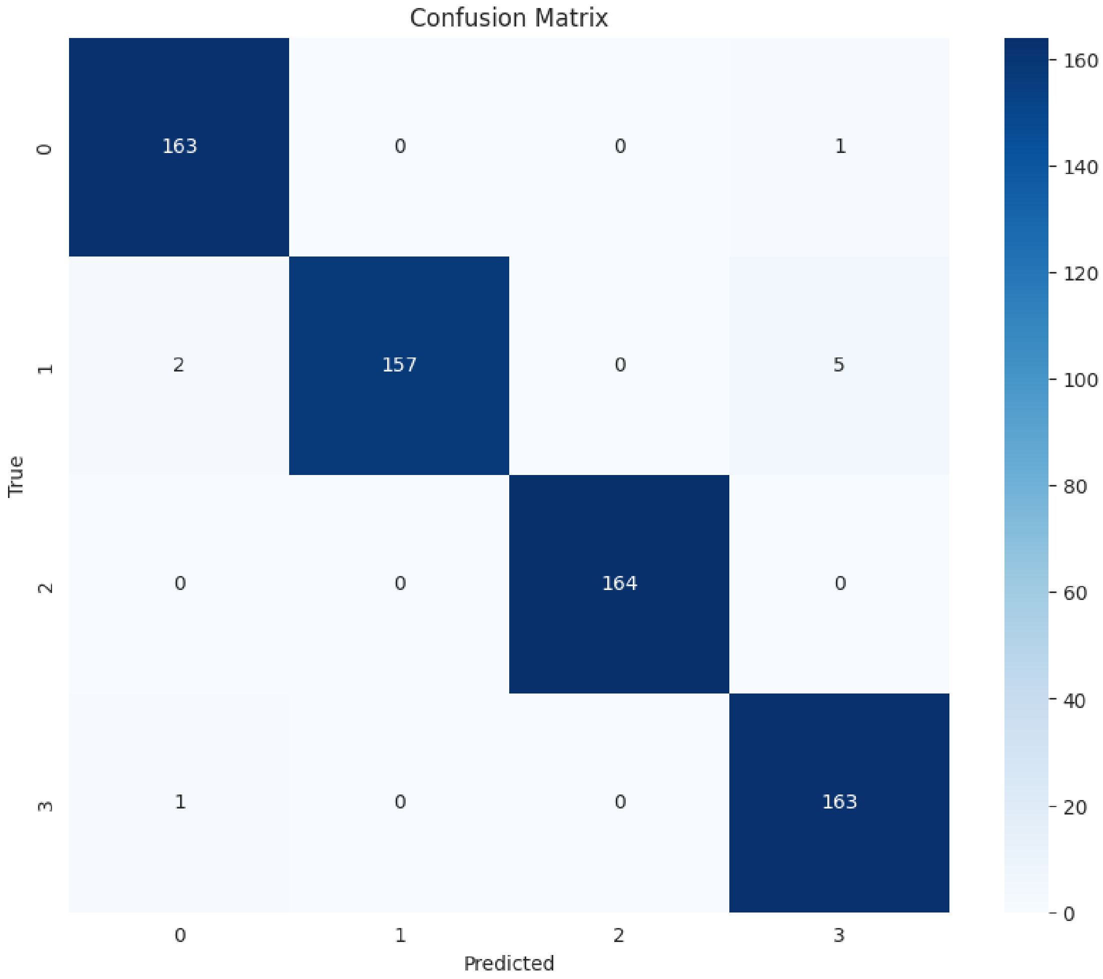
```
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_gen_new.classes
```

```
41/41 ━━━━━━━━━━━━━━━━━━━━2s 31ms/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()
```



```
from sklearn.metrics import classification_report
```

```
print("\nClassification Report:")
print(classification_report(y_true, y_pred_classes))
```

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

           0       0.98      0.99      0.99       164
           1       1.00      0.96      0.98       164
           2       1.00      1.00      1.00       164
           3       0.96      0.99      0.98       164

    accuracy                           0.99       656
   macro avg       0.99      0.99      0.99       656
weighted avg       0.99      0.99      0.99       656
```