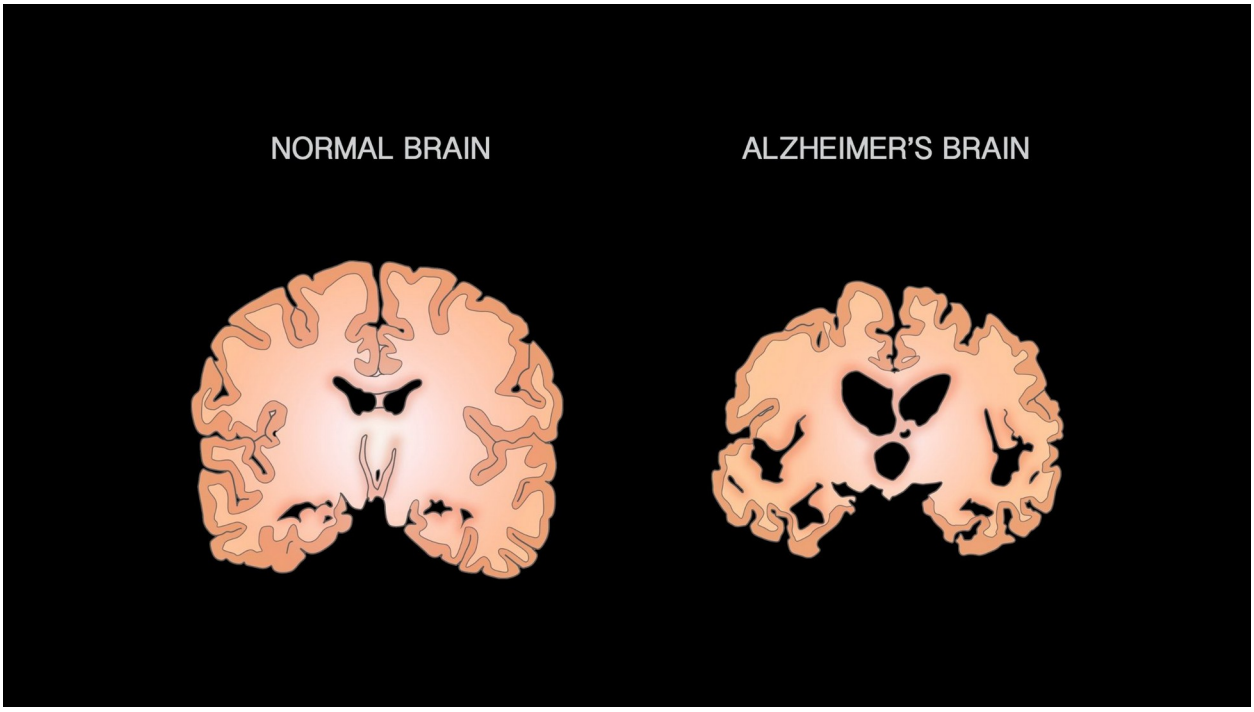
Alzheimer Disease Detection using Novel Continuous Neural Network




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

base_path = "/kaggle/input/iraqiad/IraqiAD/IraqiAD"
categories = ["AD", "CN"]

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/iraqiad/IraqiAD/IraqiAD/AD/AD_13...    AD
1  /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_28...    AD
2  /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_19...    AD
3  /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_05...    AD
4  /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_13...    AD
```

```
df.tail()
```

```
                                          image_path label
6800  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_20...    CN
6801  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_28...    CN
6802  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_11...    CN
6803  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_18...    CN
6804  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_24...    CN
```

```
df.shape
```

```
(6805, 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: 6805 entries, 0 to 6804
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  6805 non-null   object
 1   label       6805 non-null   object
dtypes: object(2)
memory usage: 106.5+ KB
```

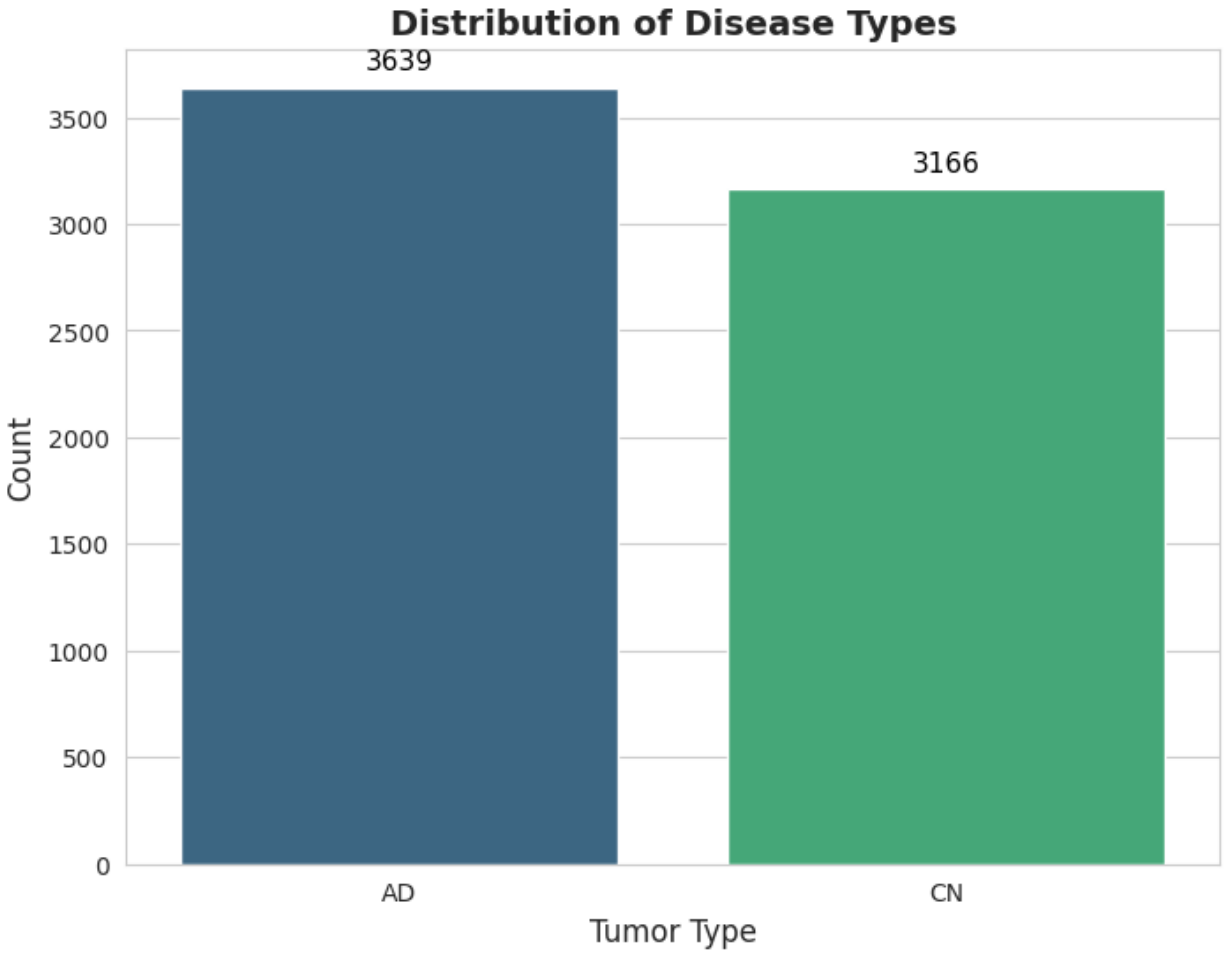
```
df['label'].unique()
```

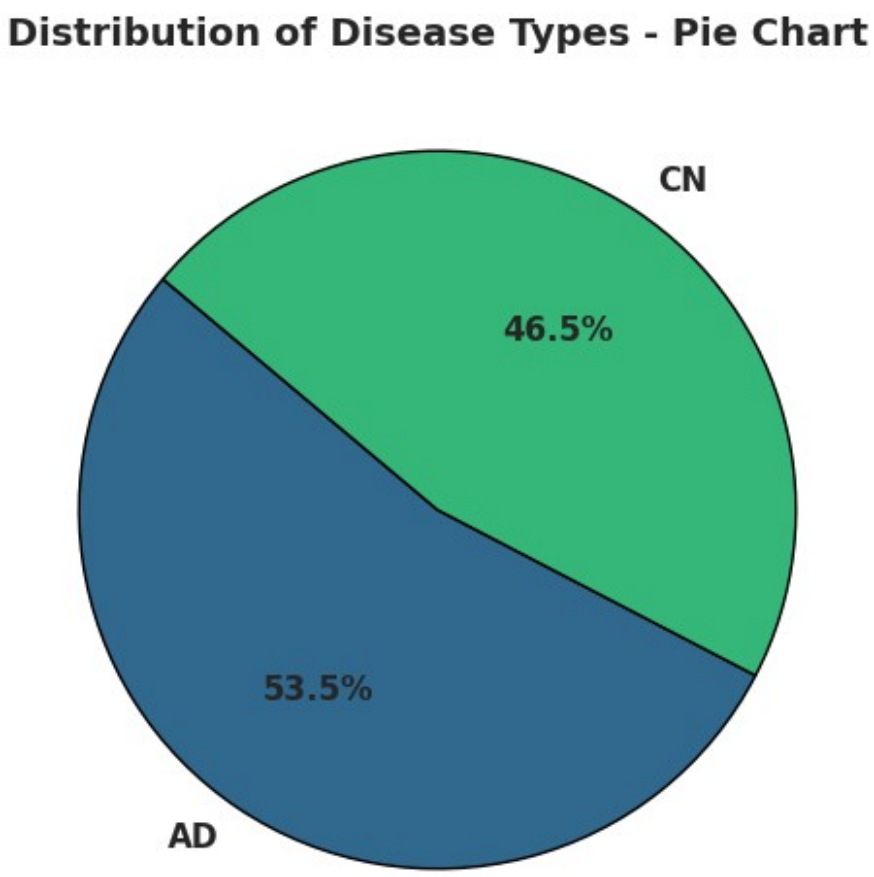
```
array(['AD', 'CN'], dtype=object)
```

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

```
label
AD    3639
CN    3166
Name: count, dtype: int64
```

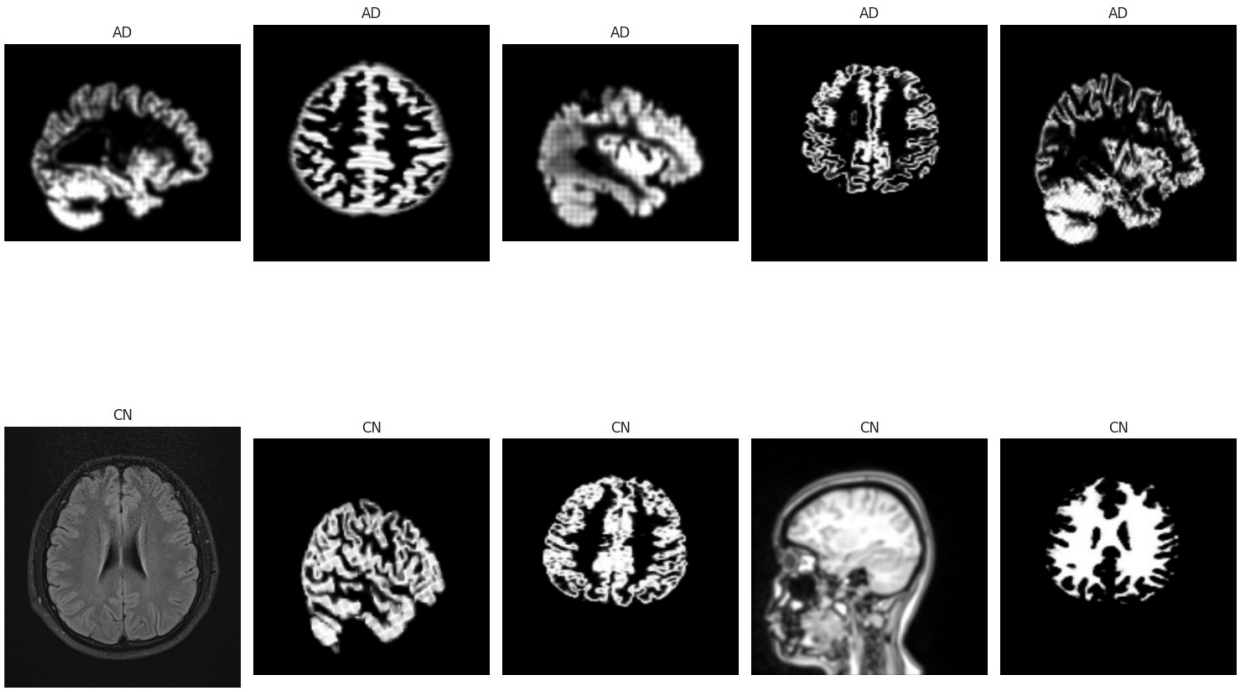
```
import seaborn as sns
import matplotlib.pyplot as plt

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 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.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
3639
3166
3500
3000
2500
2000
1500
1000
500
0
Count
AD
CN
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']]

min_samples = df['category_encoded'].value_counts().min()
balanced_df = df.groupby('category_encoded').sample(n=min_samples,
random_state=42)
balanced_df = balanced_df.reset_index(drop=True)
balanced_df = balanced_df[['image_path', 'category_encoded']]
print(balanced_df)
```

```
                                          image_path
category_encoded
0     /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_01...
0
1     /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_17...
0
2     /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_30...
0
3     /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_07...
0
4     /kaggle/input/iraqiad/IraqiAD/IraqiAD/AD/AD_36...
0
```

```
...                                                   ...
...
6327  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_18...
1
6328  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_31...
1
6329  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_00...
1
6330  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_00...
1
6331  /kaggle/input/iraqiad/IraqiAD/IraqiAD/CN/CN_09...
1

[6332 rows x 2 columns]
```

```
df_resampled = balanced_df
```

```
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')
```

```
2025-05-30 06:58:40.528470: 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 00:00:1748588320.755575      35 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1748588320.819813      35 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered

check
```

```
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']
)

import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import pandas as pd

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='binary',
    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='binary',
    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='binary',
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size
)

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.bce = tf.keras.losses.BinaryCrossentropy()

    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.bce(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(1, activation='sigmoid')(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=3,
    verbose=1
)
```

```
Found 5065 validated image filenames belonging to 2 classes.
Found 633 validated image filenames belonging to 2 classes.
Found 634 validated image filenames belonging to 2 classes.
Epoch 1/3

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1748588988.971755     131 service.cc:148] XLA service
0x7d26f400a1a0 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1748588988.972719     131 service.cc:156]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1748588988.972745     131 service.cc:156]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1748588989.482785     131 cuda_dnn.cc:529] Loaded cuDNN
version 90300
```

```
  4/317 ━━━━━━━━━━━━━━━━━━━━ 17s 55ms/step - accuracy: 0.4935 - loss:
3.5424

I0000 00:00:1748588994.034201     131 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

317/317 ━━━━━━━━━━━━━━━━━━━━ 39s 94ms/step - accuracy: 0.7043 - loss:
0.9380 - val_accuracy: 0.9415 - val_loss: 0.1666
Epoch 2/3
317/317 ━━━━━━━━━━━━━━━━━━━━ 12s 37ms/step - accuracy: 0.9563 - loss:
0.1239 - val_accuracy: 0.9684 - val_loss: 0.0865
Epoch 3/3
317/317 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9784 - loss:
0.0584 - val_accuracy: 0.9716 - val_loss: 0.0798
```

```
model.summary()
```

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
continuous_layer_3 (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_4 (Dense)	(None, 128)	25,690,240

```
| dropout_2 (Dropout)                     | (None, 128)                    |
0 |
├─────────────────────────────────────────┼────────────────────────────────┼
──────────────────┤
| dense_5 (Dense)                         | (None, 1)                      |
129 |
└─────────────────────────────────────────┴────────────────────────────────┴
──────────────────┘

 Total params: 77,074,799 (294.02 MB)

 Trainable params: 25,691,599 (98.01 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 51,383,200 (196.01 MB)

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

40/40 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - accuracy: 0.9782 - loss:
0.0566
Test Loss: 0.0663, Test Accuracy: 0.9685

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()
```


```
from sklearn.metrics import confusion_matrix
import numpy as np

y_pred = model.predict(test_gen_new)
y_pred_binary = (y_pred > 0.5).astype(int)
y_true = test_gen_new.classes
cm = confusion_matrix(y_true, y_pred_binary).ravel()
print("Confusion Matrix (TN, FP, FN, TP):", cm)
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step
Confusion Matrix (TN, FP, FN, TP): [311   6  14 303]
```

```
class_names = list(test_gen_new.class_indices.keys())
print("\nClassification Report:")
print(classification_report(y_true, y_pred_binary,
target_names=class_names))
```

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

           0       0.96      0.98      0.97       317
           1       0.98      0.96      0.97       317

    accuracy                           0.97       634
   macro avg       0.97      0.97      0.97       634
weighted avg       0.97      0.97      0.97       634
```