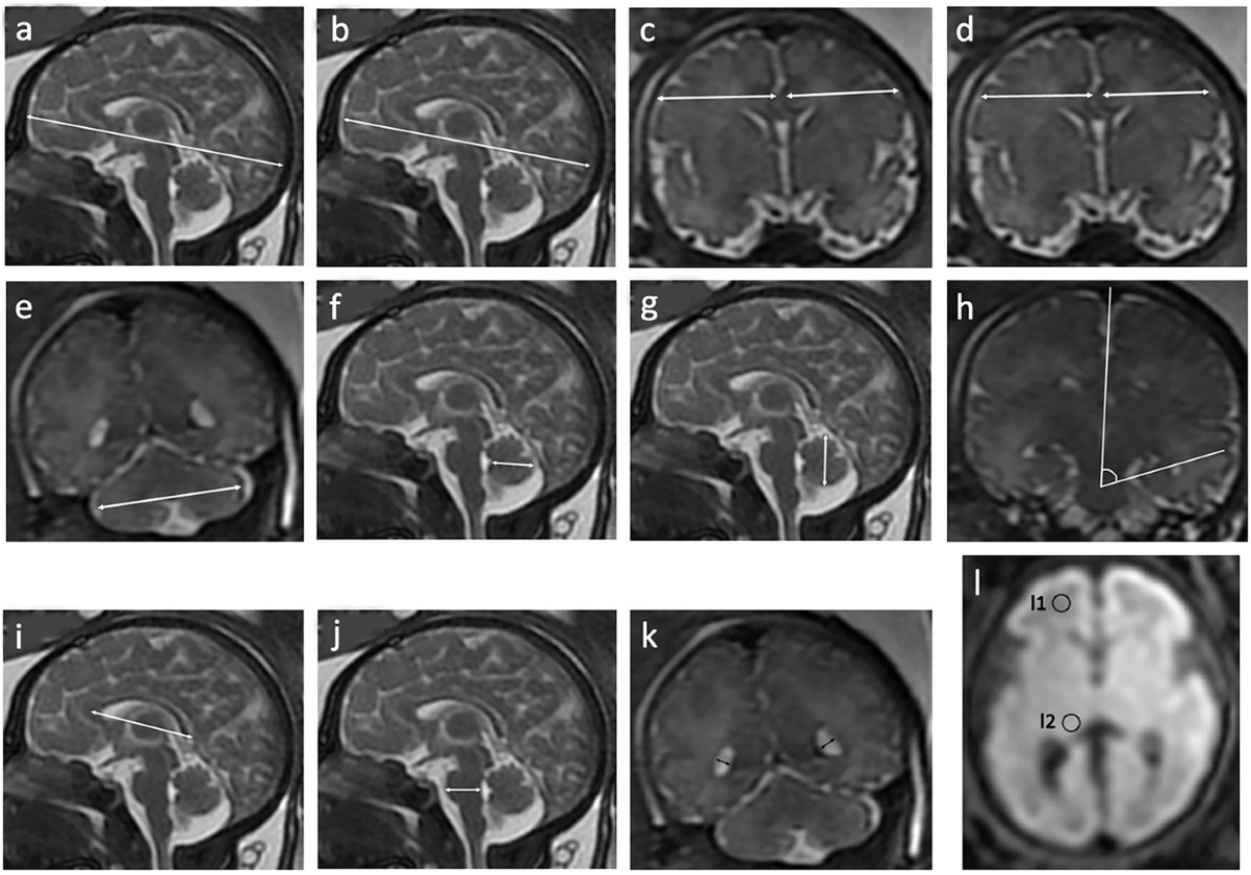
Two-Stream CNN with Transformer Attention for PCOS Image Classification




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

base_path = "/kaggle/input/pcos-xai-ultrasound-dataset/PCOS/"
categories = ["infected", "noninfected"]

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/pcos-xai-ultrasound-dataset/PCOS...	infected
1	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	infected
2	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	infected
3	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	infected
4	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	infected

```
df.tail()
```

	image_path	label
11779	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	noninfected
11780	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	noninfected
11781	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	noninfected
11782	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	noninfected
11783	/kaggle/input/pcos-xai-ultrasound-dataset/PCOS...	noninfected

```
df.shape
```

```
(11784, 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: 11784 entries, 0 to 11783
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  11784 non-null  object
 1   label       11784 non-null  object
dtypes: object(2)
memory usage: 184.3+ KB
```

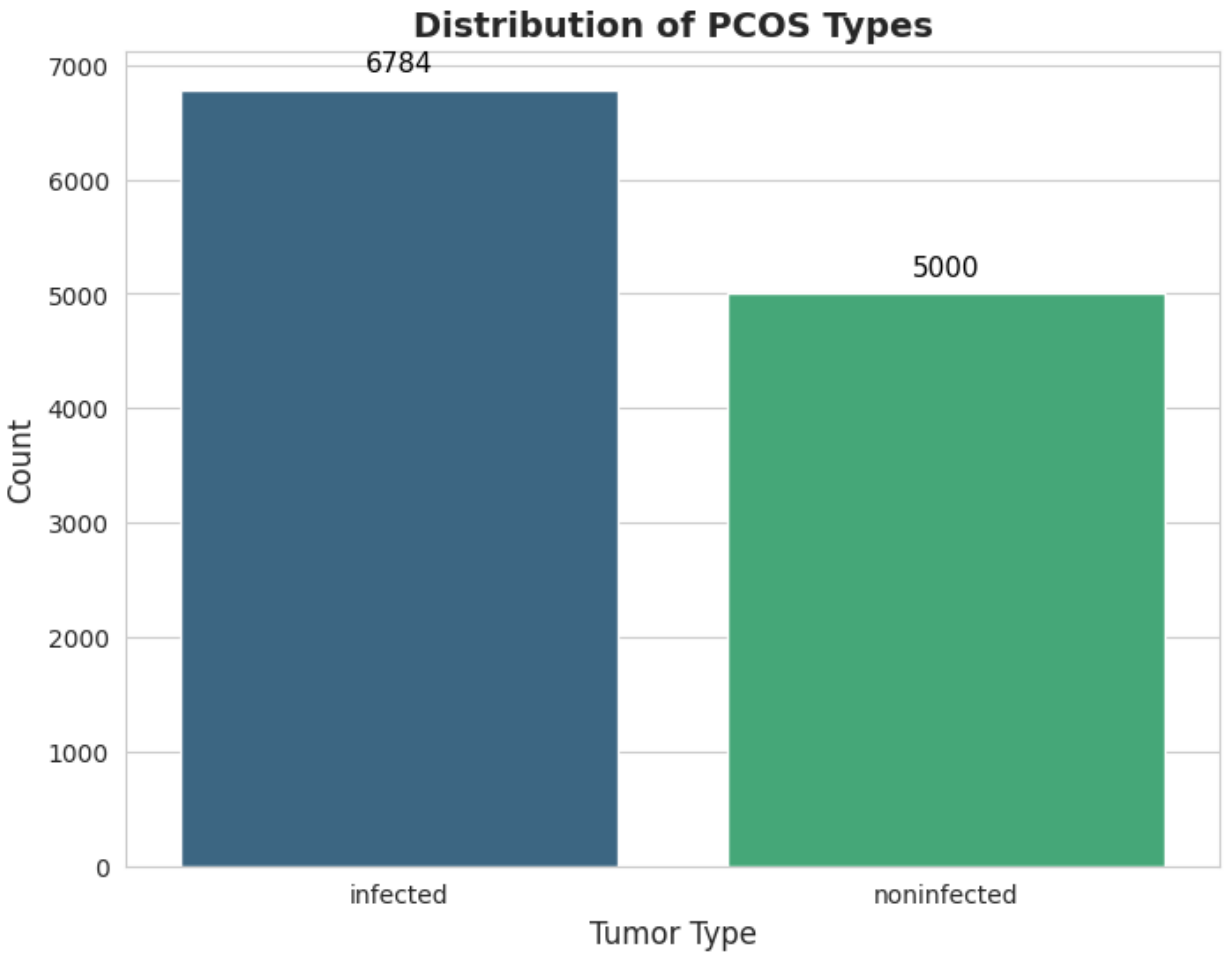
```
df['label'].unique()
```

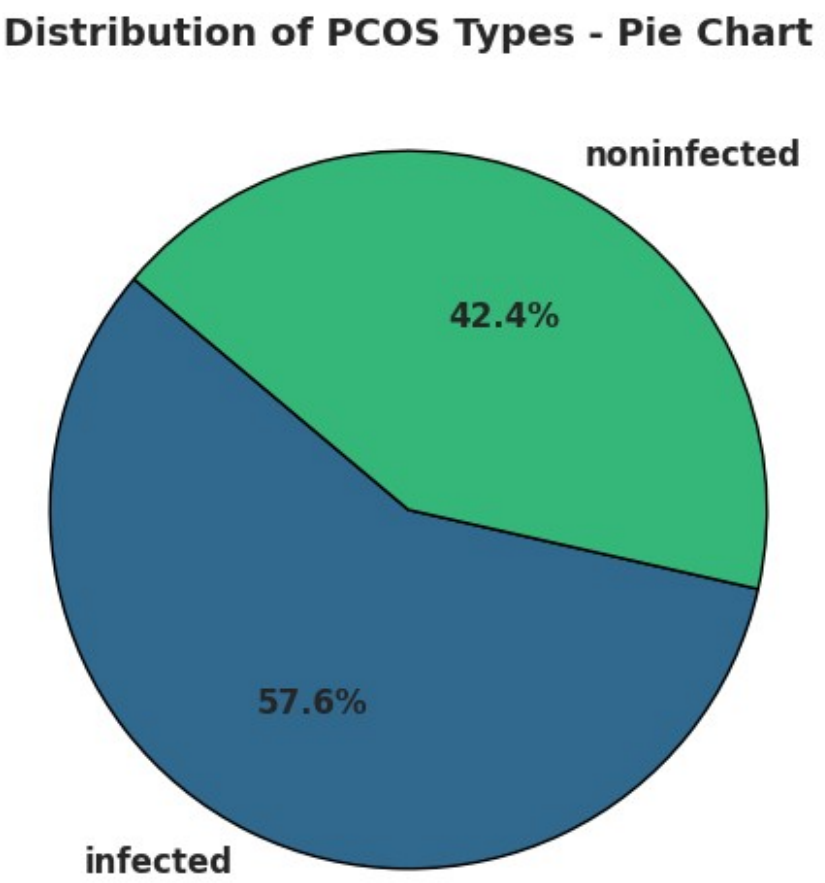
```
array(['infected', 'noninfected'], dtype=object)
```

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

```
label
infected       6784
noninfected    5000
Name: count, dtype: int64
```

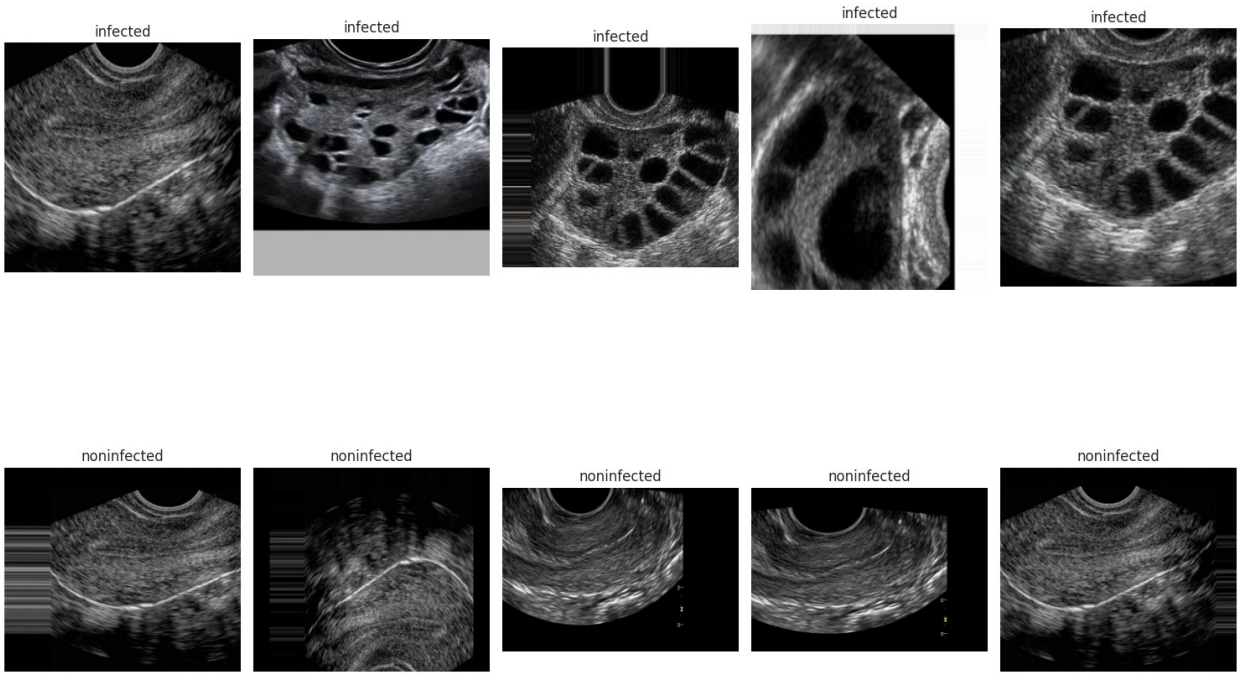
```
import seaborn as sns
import matplotlib.pyplot as plt

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 PCOS 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 PCOS Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of PCOS Types
6784
5000
7000
6000
5000
4000
3000
2000
1000
0
Count
infected
noninfected
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
0    6784
1    6784
Name: count, dtype: int64
```

```
df_resampled = df_balanced

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

import time
import shutil
import pathlib
import itertools
from PIL import Image

import cv2
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
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')
```

```
2025-05-27 06:08:41.921688: 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 00:00:1748326121.943676     275 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1748326121.950437     275 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']
)

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='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
)
```

```
Found 10854 validated image filenames belonging to 2 classes.
Found 1357 validated image filenames belonging to 2 classes.
Found 1357 validated image filenames belonging to 2 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
```

```
import tensorflow as tf
from tensorflow.keras.layers import (Input, Lambda, Conv2D,
MaxPooling2D, Flatten,
                                     Dense, Add, Reshape,
MultiHeadAttention, Concatenate, GlobalAveragePooling1D)
from tensorflow.keras.models import Model

img_size = (224, 224)
channels = 3
img_shape = (224, 224, 3)
num_classes = len(train_df_new['category_encoded'].unique())

def split_image(image):
    upper_half = image[:, :img_size[0]//2, :, :]
    lower_half = image[:, img_size[0]//2:, :, :]
    return upper_half, lower_half

def flip_lower_half(lower_half):
    return tf.image.flip_left_right(lower_half)
```

```
input_layer = Input(shape=img_shape)

upper_half, lower_half = Lambda(split_image)(input_layer)
lower_half_flipped = Lambda(flip_lower_half)(lower_half)

upper_conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')
(upper_half)
upper_pool1 = MaxPooling2D((2, 2))(upper_conv1)
upper_conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')
(upper_pool1)
upper_pool2 = MaxPooling2D((2, 2))(upper_conv2)
upper_conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')
(upper_pool2)
upper_pool3 = MaxPooling2D((2, 2))(upper_conv3)
upper_flat = Flatten()(upper_pool3)

lower_conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')
(lower_half_flipped)
lower_pool1 = MaxPooling2D((2, 2))(lower_conv1)
lower_conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')
(lower_pool1)
lower_pool2 = MaxPooling2D((2, 2))(lower_conv2)
lower_conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')
(lower_pool2)
lower_pool3 = MaxPooling2D((2, 2))(lower_conv3)
lower_flat = Flatten()(lower_pool3)

upper_dense = Dense(512, activation='relu')(upper_flat)
lower_dense = Dense(512, activation='relu')(lower_flat)

upper_reshape = Reshape((1, 512))(upper_dense)
lower_reshape = Reshape((1, 512))(lower_dense)

concat_seq = Concatenate(axis=1)([upper_reshape, lower_reshape])
attention_output = MultiHeadAttention(num_heads=4, key_dim=64)
(concat_seq, concat_seq)

attention_pooled = GlobalAveragePooling1D()(attention_output)

fc1 = Dense(256, activation='relu')(attention_pooled)
fc2 = Dense(128, activation='relu')(fc1)
output = Dense(num_classes, activation='softmax')(fc2)

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

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.summary()
```

```
I0000 00:00:1748326125.377782     275 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB
memory:  -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0,
compute capability: 7.5
I0000 00:00:1748326125.378426     275 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB
memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0,
compute capability: 7.5
```

```
Model: "functional"
```

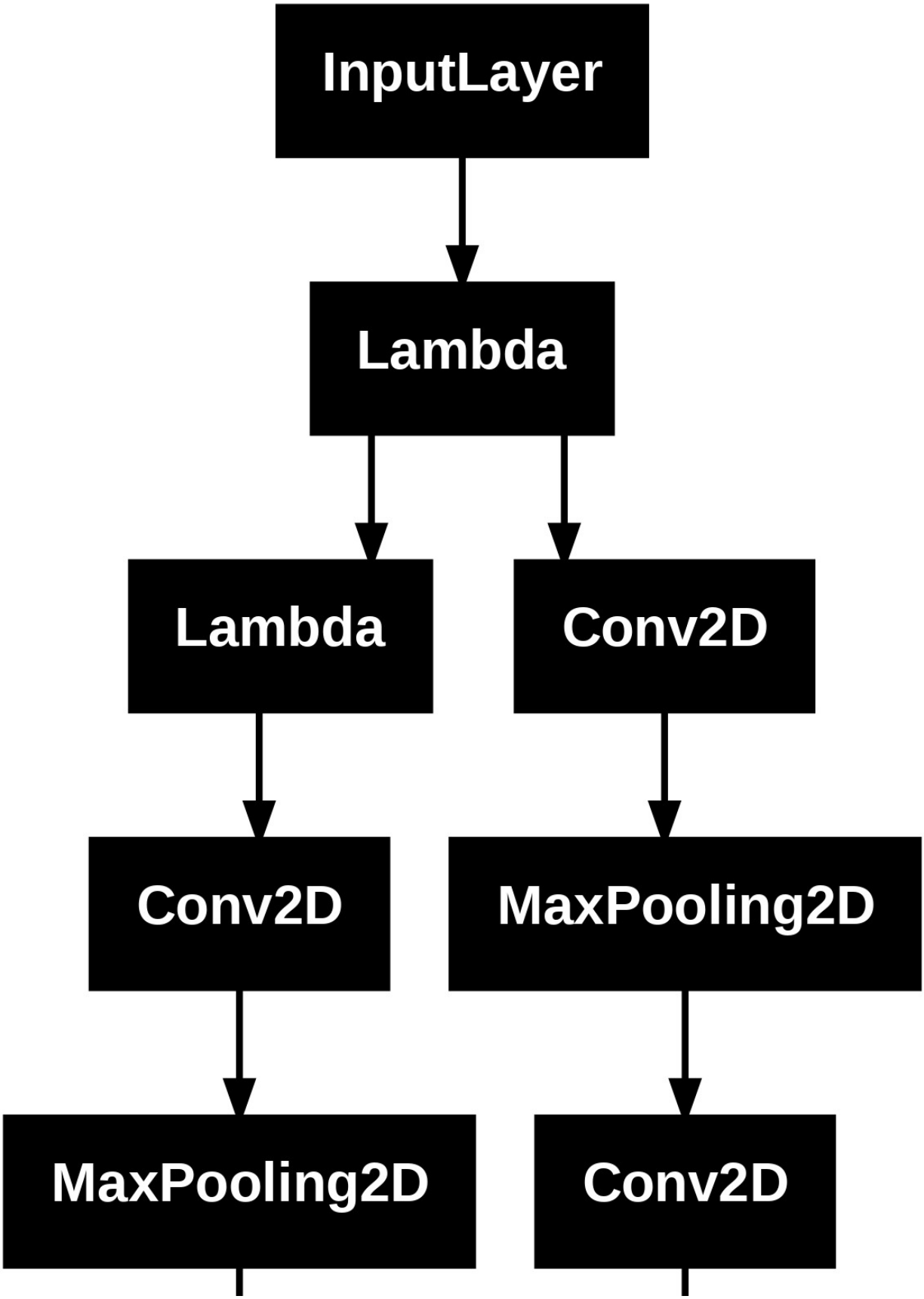
```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape                           ┃                 Param # ┃
┃ Connected to                    ┃                                        ┃                         ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)        │ (None, 224, 224, 3)                    │                       0 │
│ -                               │                                        │                         │
├─────────────────────────────────┼────────────────────────────────────────┼─────────────────────────┤
│ lambda (Lambda)                 │ [(None, 112, 224, 3),                  │                       0 │
│ input_layer[0][0]               │                                        │                         │
│                                 │ (None, 112, 224, 3)]                   │                         │
│                                 │                                        │                         │
├─────────────────────────────────┼────────────────────────────────────────┼─────────────────────────┤
│ lambda_1 (Lambda)               │ (None, 112, 224, 3)                    │                       0 │
│ lambda[0][1]                    │                                        │                         │
├─────────────────────────────────┼────────────────────────────────────────┼─────────────────────────┤
│ conv2d (Conv2D)                 │ (None, 112, 224, 32)                   │                     896 │
│ lambda[0][0]                    │                                        │                         │
├─────────────────────────────────┼────────────────────────────────────────┼─────────────────────────┤
│ conv2d_3 (Conv2D)               │ (None, 112, 224, 32)                   │                     896 │
│ lambda_1[0][0]                  │                                        │                         │
├─────────────────────────────────┼────────────────────────────────────────┼─────────────────────────┤
│ max_pooling2d                   │ (None, 56, 112, 32)                    │                       0 │
│ conv2d[0][0]                    │                                        │                         │
│ (MaxPooling2D)                  │                                        │                         │
│                                 │                                        │                         │
├─────────────────────────────────┼────────────────────────────────────────┼─────────────────────────┤
│ max_pooling2d_3                 │ (None, 56, 112, 32)                    │                       0 │
│ conv2d_3[0][0]                  │                                        │                         │
```

```
(MaxPooling2D)                  |                           |
                                |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
conv2d_1 (Conv2D)               | (None, 56, 112, 64)       |          18,496
max_pooling2d[0][0]             |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
conv2d_4 (Conv2D)               | (None, 56, 112, 64)       |          18,496
max_pooling2d_3[0][0]           |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
max_pooling2d_1                 | (None, 28, 56, 64)        |               0
conv2d_1[0][0]                  |                           |
(MaxPooling2D)                  |                           |
                                |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
max_pooling2d_4                 | (None, 28, 56, 64)        |               0
conv2d_4[0][0]                  |                           |
(MaxPooling2D)                  |                           |
                                |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
conv2d_2 (Conv2D)               | (None, 28, 56, 128)       |          73,856
max_pooling2d_1[0][0]           |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
conv2d_5 (Conv2D)               | (None, 28, 56, 128)       |          73,856
max_pooling2d_4[0][0]           |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
max_pooling2d_2                 | (None, 14, 28, 128)       |               0
conv2d_2[0][0]                  |                           |
(MaxPooling2D)                  |                           |
                                |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
max_pooling2d_5                 | (None, 14, 28, 128)       |               0
conv2d_5[0][0]                  |                           |
(MaxPooling2D)                  |                           |
                                |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
flatten (Flatten)               | (None, 50176)             |               0
max_pooling2d_2[0][0]           |                           |
--------------------------------+---------------------------+-----------------
--------------------------------+
flatten_1 (Flatten)             | (None, 50176)             |               0
```

max_pooling2d_5[0][0]		
dense (Dense) flatten[0][0]	(None, 512)	25,690,624
dense_1 (Dense) flatten_1[0][0]	(None, 512)	25,690,624
reshape (Reshape) dense[0][0]	(None, 1, 512)	0
reshape_1 (Reshape) dense_1[0][0]	(None, 1, 512)	0
concatenate (Concatenate) reshape[0][0], reshape_1[0][0]	(None, 2, 512)	0
multi_head_attention concatenate[0][0], (MultiHeadAttention) concatenate[0][0]	(None, 2, 512)	525,568
global_average_pooling1d multi_head_attention[… (GlobalAveragePooling1D)	(None, 512)	0
dense_2 (Dense) global_average_poolin…	(None, 256)	131,328
dense_3 (Dense) dense_2[0][0]	(None, 128)	32,896
dense_4 (Dense) dense_3[0][0]	(None, 2)	258

```
 Total params: 52,257,794 (199.35 MB)

 Trainable params: 52,257,794 (199.35 MB)

 Non-trainable params: 0 (0.00 B)
```

```
from tensorflow.keras.utils import plot_model

plot_model(model,
           to_file='/kaggle/working/model.png',
           show_shapes=False,
           show_dtype=False,
           show_layer_names=False,
           dpi=300)
```


InputLayer
Lambda
Lambda
Conv2D
Conv2D
MaxPooling2D
MaxPooling2D
Conv2D

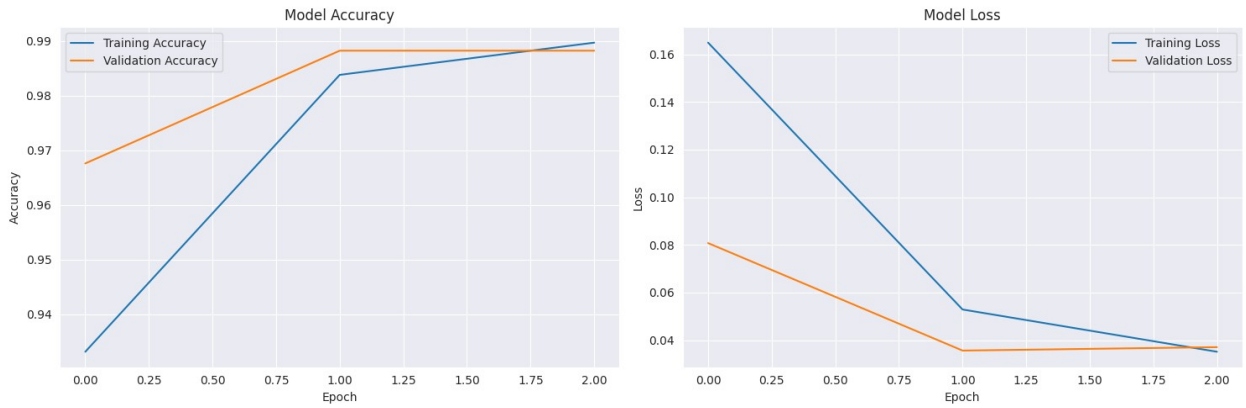
```
history = model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
    epochs=3,
    batch_size=batch_size
)
```

```
Epoch 1/3
```

```
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1748326132.102246     330 service.cc:148] XLA service
0x7f387c002eb0 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1748326132.102301     330 service.cc:156]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1748326132.102306     330 service.cc:156]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1748326132.661086     330 cuda_dnn.cc:529] Loaded cuDNN
version 90300

  3/679 ━━━━━━━━━━━━━━━━━━━━ 34s 51ms/step - accuracy: 0.4514 - loss:
0.7292

I0000 00:00:1748326136.365558     330 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

679/679 ━━━━━━━━━━━━━━━━━━━━ 54s 65ms/step - accuracy: 0.8683 - loss:
0.2710 - val_accuracy: 0.9676 - val_loss: 0.0807
Epoch 2/3
679/679 ━━━━━━━━━━━━━━━━━━━━ 38s 55ms/step - accuracy: 0.9815 - loss:
0.0573 - val_accuracy: 0.9882 - val_loss: 0.0356
Epoch 3/3
679/679 ━━━━━━━━━━━━━━━━━━━━ 36s 53ms/step - accuracy: 0.9885 - loss:
0.0369 - val_accuracy: 0.9882 - val_loss: 0.0371
```

```
def plot_training_history(history):

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    ax1.plot(history.history['accuracy'], label='Training Accuracy')
    ax1.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    ax1.set_title('Model Accuracy')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()
    ax1.grid(True)

    ax2.plot(history.history['loss'], label='Training Loss')
```

```
    ax2.plot(history.history['val_loss'], label='Validation Loss')
    ax2.set_title('Model Loss')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True)

    plt.tight_layout()
    plt.show()

plot_training_history(history)
```



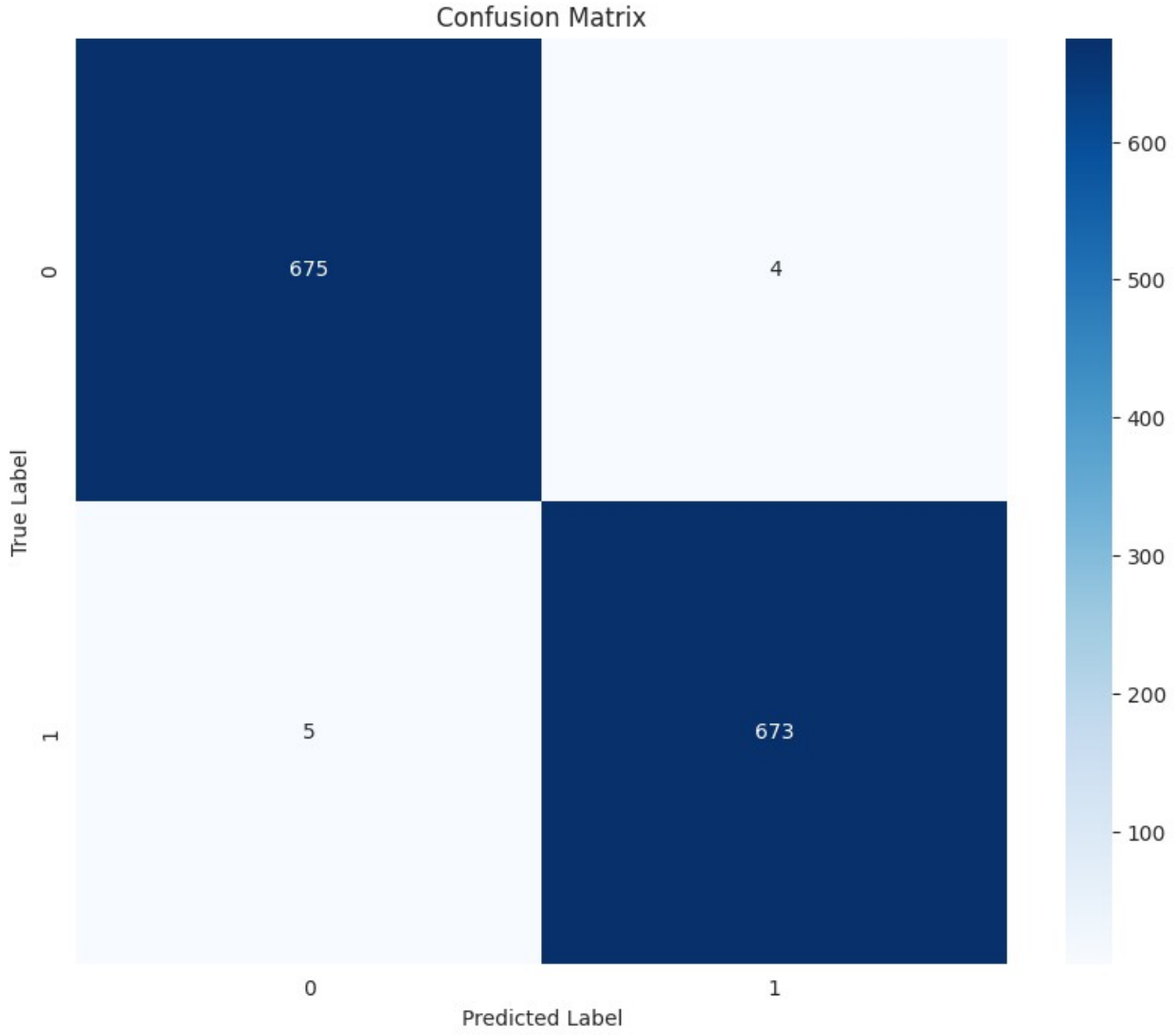

```
test_loss, test_accuracy = model.evaluate(test_gen_new)
print(f"Test Accuracy: {test_accuracy:.4f}, Test Loss:
{test_loss:.4f}")
```

```
85/85 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - accuracy: 0.9961 - loss:
0.0149
Test Accuracy: 0.9934, Test Loss: 0.0202
```

```
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

test_gen_new.reset()
y_pred_probs = model.predict(test_gen_new, verbose=1)
y_pred_classes = np.argmax(y_pred_probs, axis=1)

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

cm = confusion_matrix(y_true, y_pred_classes)

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 Label')
plt.ylabel('True Label')
plt.show()

print("Classification Report:\n")
print(classification_report(y_true, y_pred_classes,
target_names=class_names))
```

```
85/85 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step
```




```
Classification Report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       679
```

```
           1       0.99      0.99      0.99       678

    accuracy                           0.99      1357
   macro avg       0.99      0.99      0.99      1357
weighted avg       0.99      0.99      0.99      1357
```