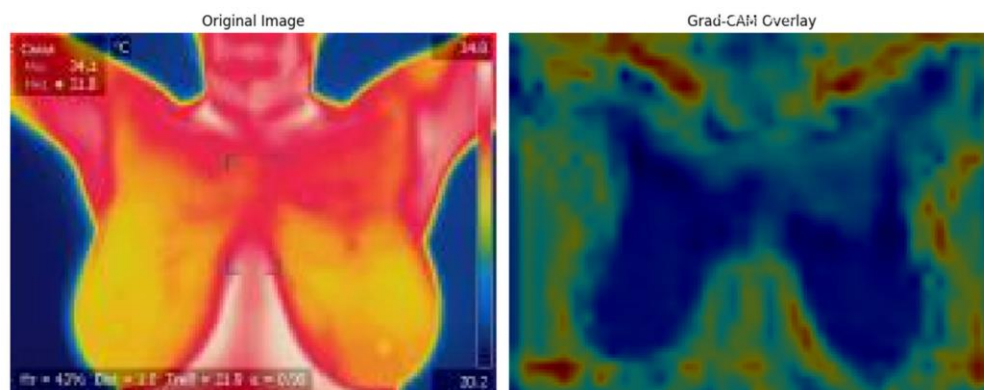


Breast cancer detection using thermography DMR-IR



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

base_path = "/kaggle/input/breast-cancer-detection-using-thermography/BCD_Dataset"
categories = ["Sick", "Unknown_class", "normal"]

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/breast-cancer-detection-using-th...  Sick
1  /kaggle/input/breast-cancer-detection-using-th...  Sick
2  /kaggle/input/breast-cancer-detection-using-th...  Sick
3  /kaggle/input/breast-cancer-detection-using-th...  Sick
4  /kaggle/input/breast-cancer-detection-using-th...  Sick

df.tail()
```

```

                                image_path  label
357 /kaggle/input/breast-cancer-detection-using-th... normal
358 /kaggle/input/breast-cancer-detection-using-th... normal
359 /kaggle/input/breast-cancer-detection-using-th... normal
360 /kaggle/input/breast-cancer-detection-using-th... normal
361 /kaggle/input/breast-cancer-detection-using-th... normal

```

```
df.shape
```

```
(362, 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: 362 entries, 0 to 361
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   image_path  362 non-null   object
 1   label       362 non-null   object
dtypes: object(2)
memory usage: 5.8+ KB

```

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

```
array(['Sick', 'Unknown_class', 'normal'], dtype=object)
```

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

```

label
normal      162
Sick        100
Unknown_class 100
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 Images", 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()

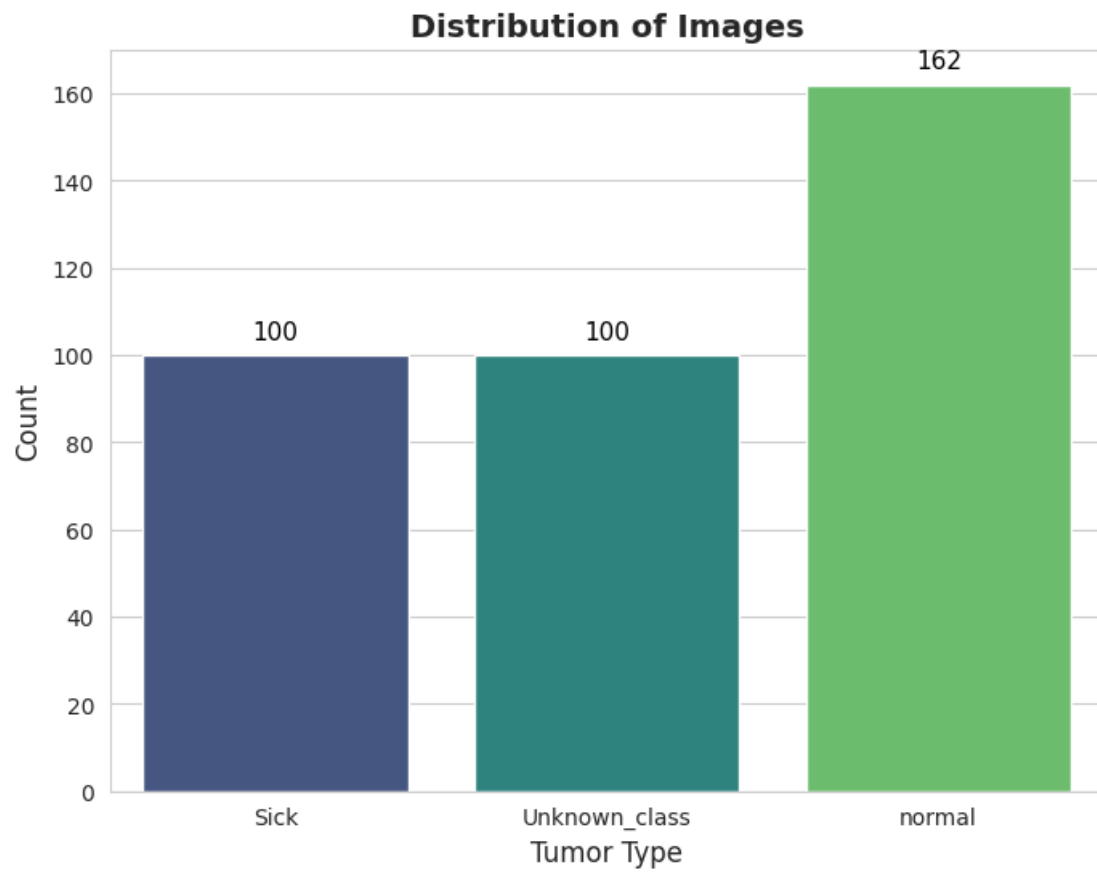
fig, ax = plt.subplots(figsize=(8, 6))
colors = sns.color_palette("viridis", len(label_counts))

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})

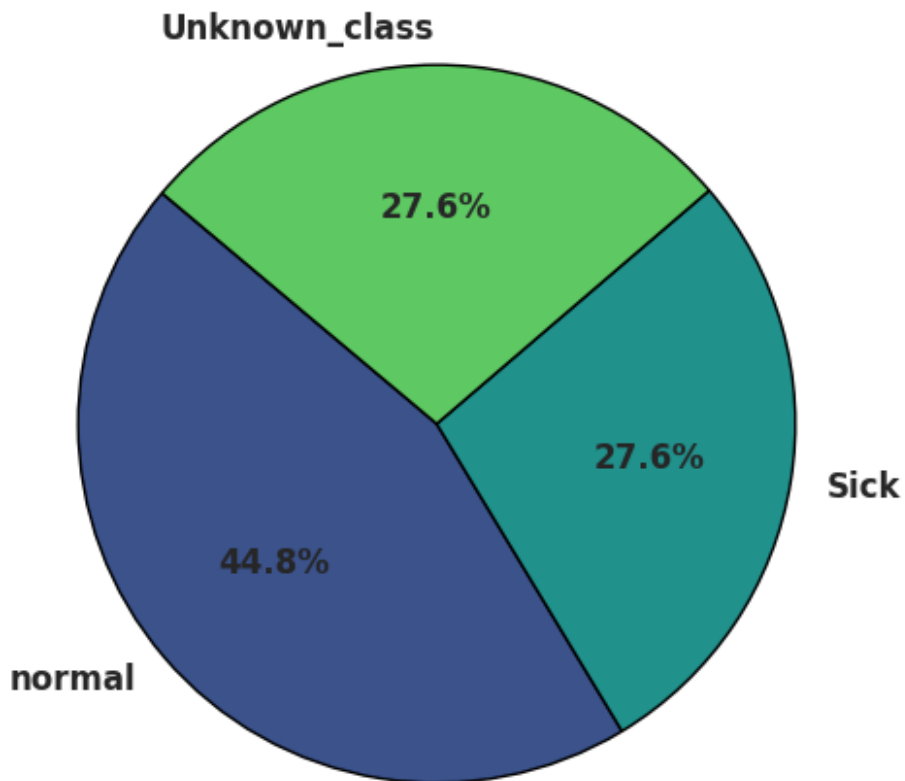
ax.set_title("Distribution of Images - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()

```



Distribution of Images - Pie Chart



```
import cv2

num_images = 5

plt.figure(figsize=(15, 12))

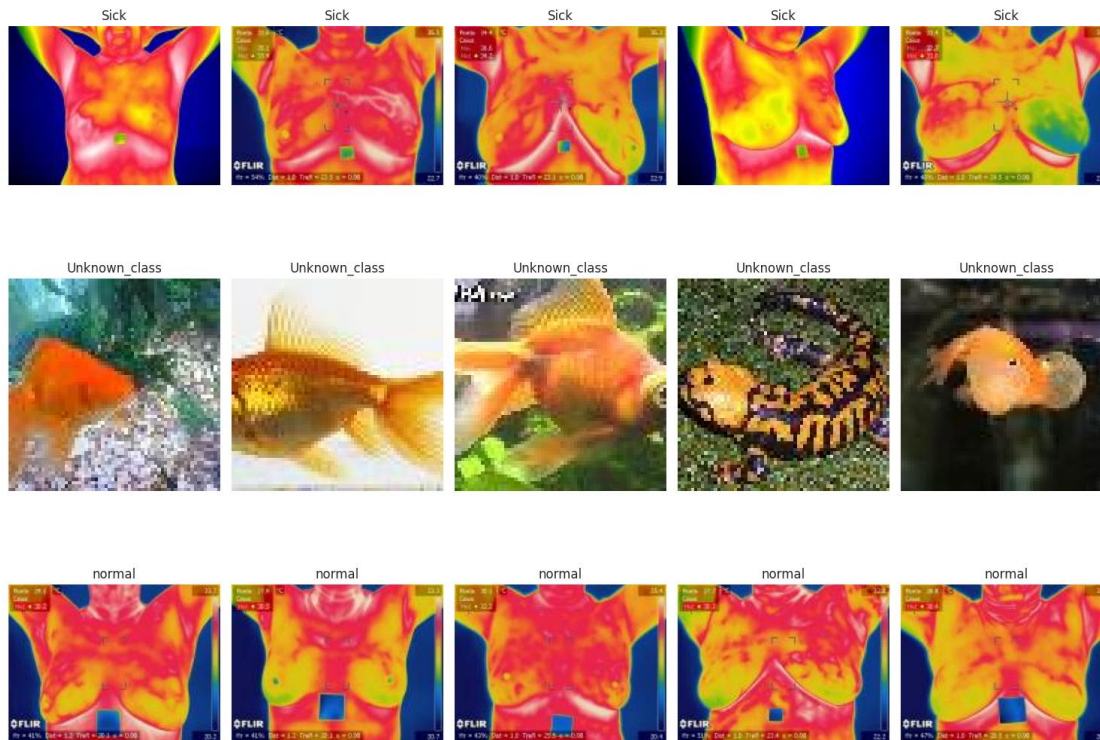
for i, category in enumerate(categories):
    category_images = df[df['label'] ==
category]['image_path'].iloc[:num_images]

    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']]

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

majority_class = df[df['category_encoded'] ==
df['category_encoded'].value_counts().idxmax()]
minority_class = df[df['category_encoded'] !=
df['category_encoded'].value_counts().idxmax()]

oversampled_minority_class = minority_class.sample(n=majority_class_count,
replace=True, random_state=42)

balanced_df = pd.concat([majority_class, oversampled_minority_class])

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/breast-cancer-detection-using-th...      2
```

```

1    /kaggle/input/breast-cancer-detection-using-th...      2
2    /kaggle/input/breast-cancer-detection-using-th...      2
3    /kaggle/input/breast-cancer-detection-using-th...      2
4    /kaggle/input/breast-cancer-detection-using-th...      2
..    ...
319  /kaggle/input/breast-cancer-detection-using-th...      0
320  /kaggle/input/breast-cancer-detection-using-th...      0
321  /kaggle/input/breast-cancer-detection-using-th...      1
322  /kaggle/input/breast-cancer-detection-using-th...      0
323  /kaggle/input/breast-cancer-detection-using-th...      1

```

```
[324 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')
```

```

2025-05-06 08:35:18.967098: 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 00:00:1746520519.357888      31 cuda_dnn.cc:8310] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
E0000 00:00:1746520519.477473      31 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']
)

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 259 validated image filenames belonging to 3 classes.
 Found 32 validated image filenames belonging to 3 classes.
 Found 33 validated image filenames belonging to 3 classes.

```
import tensorflow as tf
```

```
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 import layers, Model

```

```

class SimpleAttention(layers.Layer):
    def __init__(self, **kwargs):
        super(SimpleAttention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.in_channels = input_shape[-1]
        self.H = input_shape[1]
        self.W = input_shape[2]
        self.dense = layers.Dense(1, use_bias=False)
        super(SimpleAttention, self).build(input_shape)

    def call(self, inputs):

        weights = self.dense(inputs)
        return inputs * weights

    def compute_output_shape(self, input_shape):

        return input_shape

```

```
def create_simple_cnn(input_shape=(224, 224, 3), num_classes=3):
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(32, 3, padding='same', activation='relu')(inputs)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)

    att = SimpleAttention()(x)
    x = layers.concatenate([x, att])
    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    return Model(inputs, outputs)
```

```
simple_model = create_simple_cnn(num_classes=3)
simple_model.summary()
```

```
I0000 00:00:1746520537.790241      31 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 00:00:1746520537.790909      31 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) Connected to	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0 -
conv2d (Conv2D) input_layer[0][0]	(None, 224, 224, 32)	896
max_pooling2d conv2d[0][0] (MaxPooling2D)	(None, 112, 112, 32)	0

conv2d_1 (Conv2D) max_pooling2d[0][0]	(None, 112, 112, 64)	18,496
max_pooling2d_1 conv2d_1[0][0] (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_2 (Conv2D) max_pooling2d_1[0][0]	(None, 56, 56, 64)	36,928
max_pooling2d_2 conv2d_2[0][0] (MaxPooling2D)	(None, 28, 28, 64)	0
conv2d_3 (Conv2D) max_pooling2d_2[0][0]	(None, 28, 28, 64)	36,928
max_pooling2d_3 conv2d_3[0][0] (MaxPooling2D)	(None, 14, 14, 64)	0
simple_attention max_pooling2d_3[0][0] (SimpleAttention)	(None, 14, 14, 64)	0
concatenate (Concatenate) max_pooling2d_3[0][0], simple_attention[0][0]	(None, 14, 14, 128)	0
global_average_pooling2d concatenate[0][0] (GlobalAveragePooling2D)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

global_average_poolin... |

Total params: 93,635 (365.76 KB)

Trainable params: 93,635 (365.76 KB)

Non-trainable params: 0 (0.00 B)

```
simple_model.compile(optimizer=Adam(learning_rate=0.0001),
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
```

```
history = simple_model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
    epochs=20,
    verbose=1
)
```

Epoch 1/20

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

```
I0000 00:00:1746520543.118761      96 service.cc:148] XLA service
0x78b4d8085450 initialized for platform CUDA (this does not guarantee that
XLA will be used). Devices:
I0000 00:00:1746520543.120885      96 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1746520543.120905      96 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1746520543.498021      96 cuda_dnn.cc:529] Loaded cuDNN version
90300
```

3/17 ————— 0s 60ms/step - accuracy: 0.3472 - loss: 1.1091

```
I0000 00:00:1746520547.706082      96 device_compiler.h:188] Compiled cluster
using XLA! This line is logged at most once for the lifetime of the process.
```

17/17 ————— 12s 235ms/step - accuracy: 0.3809 - loss: 1.0937 -
val_accuracy: 0.5000 - val_loss: 1.0454

Epoch 2/20

17/17 ————— 1s 37ms/step - accuracy: 0.5135 - loss: 1.0287 -
val_accuracy: 0.5000 - val_loss: 1.0113

Epoch 3/20

17/17 ————— 1s 38ms/step - accuracy: 0.5038 - loss: 0.9956 -
val_accuracy: 0.5000 - val_loss: 0.9804

Epoch 4/20

17/17 ————— 1s 39ms/step - accuracy: 0.5136 - loss: 0.9536 -
val_accuracy: 0.5000 - val_loss: 0.9221

Epoch 5/20

```

17/17 _____ 1s 43ms/step - accuracy: 0.6154 - loss: 0.8642 -
val_accuracy: 0.7188 - val_loss: 0.8146
Epoch 6/20
17/17 _____ 1s 40ms/step - accuracy: 0.6977 - loss: 0.7670 -
val_accuracy: 0.7188 - val_loss: 0.6726
Epoch 7/20
17/17 _____ 1s 37ms/step - accuracy: 0.6419 - loss: 0.6842 -
val_accuracy: 0.7188 - val_loss: 0.5700
Epoch 8/20
17/17 _____ 1s 35ms/step - accuracy: 0.7037 - loss: 0.6337 -
val_accuracy: 0.7500 - val_loss: 0.5698
Epoch 9/20
17/17 _____ 1s 35ms/step - accuracy: 0.6506 - loss: 0.6396 -
val_accuracy: 0.6875 - val_loss: 0.5768
Epoch 10/20
17/17 _____ 1s 36ms/step - accuracy: 0.6968 - loss: 0.5779 -
val_accuracy: 0.7500 - val_loss: 0.5399
Epoch 11/20
17/17 _____ 1s 34ms/step - accuracy: 0.7665 - loss: 0.5888 -
val_accuracy: 0.7812 - val_loss: 0.5265
Epoch 12/20
17/17 _____ 1s 34ms/step - accuracy: 0.7434 - loss: 0.5310 -
val_accuracy: 0.7500 - val_loss: 0.5183
Epoch 13/20
17/17 _____ 1s 35ms/step - accuracy: 0.8228 - loss: 0.5321 -
val_accuracy: 0.7500 - val_loss: 0.5201
Epoch 14/20
17/17 _____ 1s 34ms/step - accuracy: 0.7557 - loss: 0.5013 -
val_accuracy: 0.7812 - val_loss: 0.5164
Epoch 15/20
17/17 _____ 1s 35ms/step - accuracy: 0.8359 - loss: 0.4433 -
val_accuracy: 0.7812 - val_loss: 0.5075
Epoch 16/20
17/17 _____ 1s 36ms/step - accuracy: 0.8301 - loss: 0.4063 -
val_accuracy: 0.7812 - val_loss: 0.6188
Epoch 17/20
17/17 _____ 1s 38ms/step - accuracy: 0.8497 - loss: 0.4186 -
val_accuracy: 0.7812 - val_loss: 0.6249
Epoch 18/20
17/17 _____ 1s 38ms/step - accuracy: 0.8387 - loss: 0.4443 -
val_accuracy: 0.8125 - val_loss: 0.5495
Epoch 19/20
17/17 _____ 1s 32ms/step - accuracy: 0.8863 - loss: 0.3764 -
val_accuracy: 0.8125 - val_loss: 0.4936
Epoch 20/20
17/17 _____ 1s 34ms/step - accuracy: 0.8297 - loss: 0.4200 -
val_accuracy: 0.8125 - val_loss: 0.4749

```

```

def plot_history(history):
    plt.figure(figsize=(12, 4))

```

```

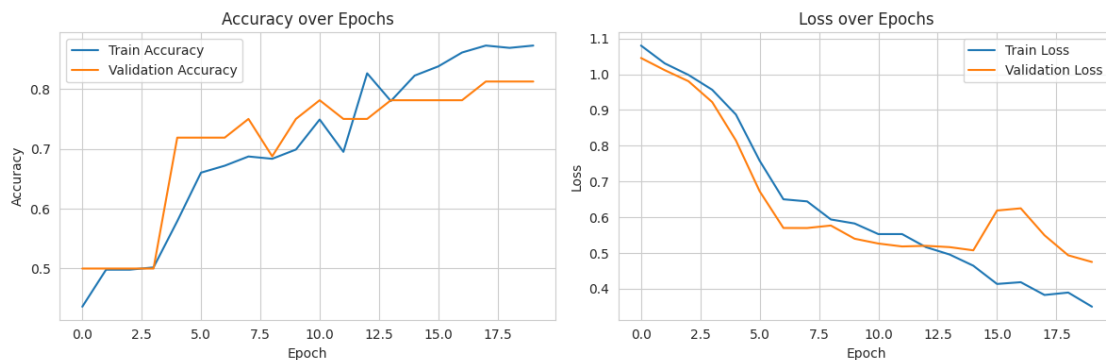
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

```

plot_history(history)



```

test_labels = test_gen_new.classes
predictions = simple_model.predict(test_gen_new)
predicted_classes = np.argmax(predictions, axis=1)

```

3/3 ————— 1s 338ms/step

```

report = classification_report(test_labels, predicted_classes,
target_names=list(test_gen_new.class_indices.keys()))
print(report)

```

	precision	recall	f1-score	support
0	1.00	0.56	0.71	9
1	1.00	1.00	1.00	7
2	0.81	1.00	0.89	17
accuracy			0.88	33

macro avg	0.94	0.85	0.87	33
weighted avg	0.90	0.88	0.87	33

```
conf_matrix = confusion_matrix(test_labels, predicted_classes)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=list(test_gen_new.class_indices.keys()),
            yticklabels=list(test_gen_new.class_indices.keys()))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

