endoscopy-wce-image-classificat-1

February 6, 2025

[ ]:

```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

[2]:

```python
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report
import cv2
import itertools
```

[3]:

```
# Set dataset path
dataset_path = "/kaggle/input/capsule-endoscopy-dataset-kauhc"  # Update with␣
 ↪actual path
```

[4]:

```
# Define image dimensions & batch size
IMG_SIZE = (128, 128)
BATCH_SIZE = 32

# Load dataset with ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training')

val_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation')
```

```
Found 2642 images belonging to 3 classes.
Found 659 images belonging to 3 classes.
```

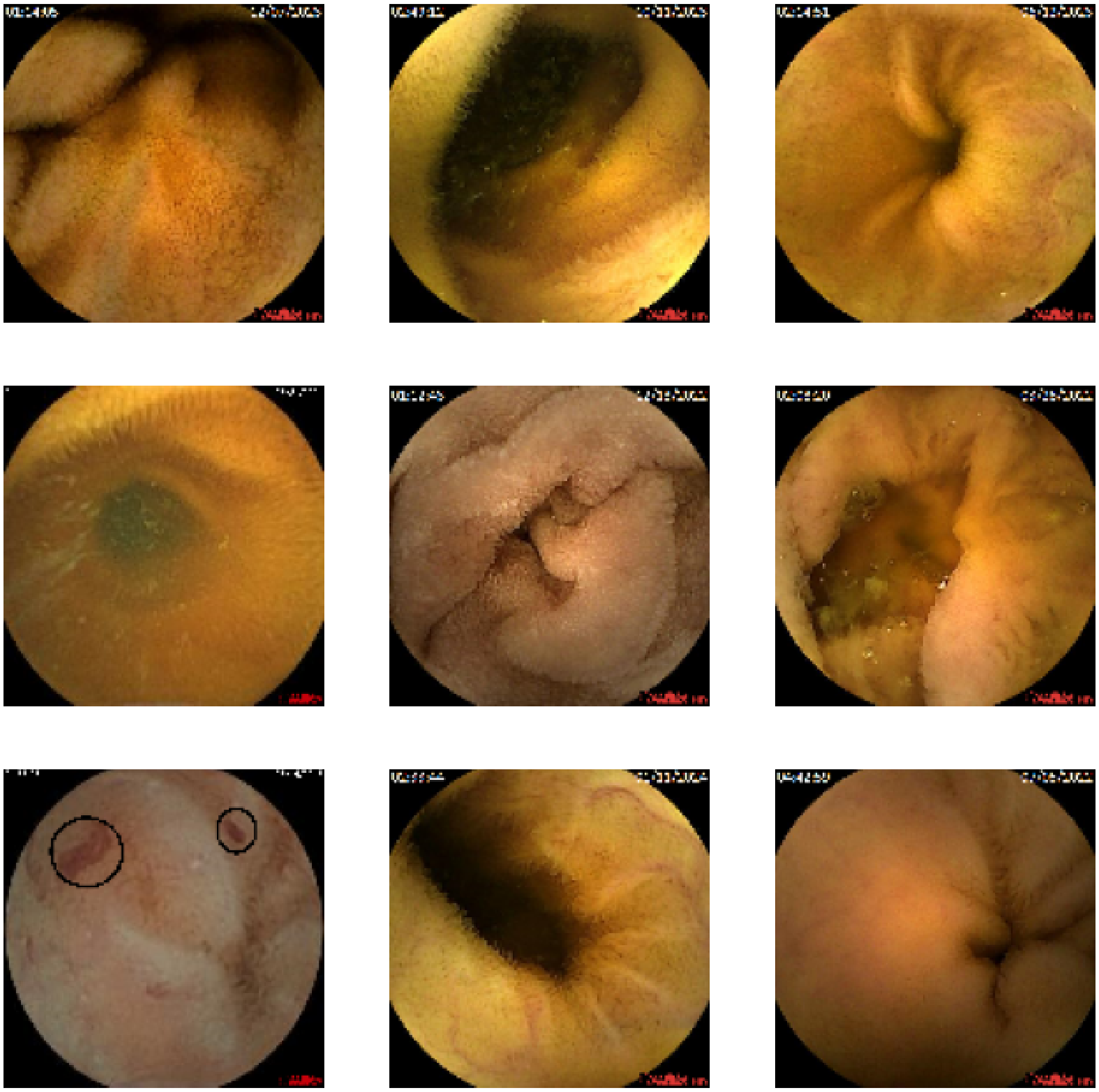
[5]:

```
# Display class labels
print("Class Labels:", train_generator.class_indices)

# Visualize 9 images from the dataset
def visualize_images(generator):
    images, labels = next(generator)
    fig, axes = plt.subplots(3, 3, figsize=(8, 8))
    axes = axes.flatten()
    for img, lbl, ax in zip(images[:9], labels[:9], axes):
        ax.imshow(img)
        ax.axis('off')
    plt.show()

visualize_images(train_generator)
```

```
Class Labels: {'AVM': 0, 'Normal': 1, 'Ulcer': 2}
```



[6]:

```
# Define CNN Model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D(2,2),
    BatchNormalization(),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    BatchNormalization(),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[7]:

```
# Compile Model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Model Summary
model.summary()
```

```
Model: "sequential"


 Layer (type)                         Output Shape                          ␣
↪Param #

 conv2d (Conv2D)                      (None, 126, 126, 32)                      ␣
↪896

 max_pooling2d (MaxPooling2D)         (None, 63, 63, 32)                        ␣
↪   0

 batch_normalization                  (None, 63, 63, 32)                        ␣
↪128
 (BatchNormalization)                                                           ␣
↪

 conv2d_1 (Conv2D)                    (None, 61, 61, 64)                     ␣
↪18,496

 max_pooling2d_1 (MaxPooling2D)       (None, 30, 30, 64)                        ␣
↪   0

 batch_normalization_1                (None, 30, 30, 64)                        ␣
↪256
 (BatchNormalization)                                                           ␣
↪

 flatten (Flatten)                    (None, 57600)                             ␣
↪   0
```

```
dense (Dense)                        (None, 128)                       ␣
↪7,372,928

dropout (Dropout)                    (None, 128)                          ␣
↪  0

dense_1 (Dense)                      (None, 3)                            ␣
↪387
```

```
Total params: 7,393,091 (28.20 MB)

Trainable params: 7,392,899 (28.20 MB)

Non-trainable params: 192 (768.00 B)
```

[11]:

```
# Train Model with Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
#model_checkpoint = ModelCheckpoint("best_model.h5", save_best_only=True)
model_checkpoint = ModelCheckpoint("best_model.keras", save_best_only=True)



history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping, model_checkpoint]
)
```

```
Epoch 1/30
83/83                 6s 64ms/step -
accuracy: 0.9399 - loss: 0.1782 - val_accuracy: 0.6586 - val_loss: 6.7876
Epoch 2/30
83/83                 5s 59ms/step -
accuracy: 0.9494 - loss: 0.1424 - val_accuracy: 0.6950 - val_loss: 4.5477
Epoch 3/30
83/83                 5s 55ms/step -
accuracy: 0.9385 - loss: 0.1259 - val_accuracy: 0.7041 - val_loss: 5.5116
Epoch 4/30
83/83                 5s 53ms/step -
accuracy: 0.9520 - loss: 0.1185 - val_accuracy: 0.5402 - val_loss: 5.8825
Epoch 5/30
83/83                 5s 52ms/step -
accuracy: 0.9582 - loss: 0.0809 - val_accuracy: 0.7496 - val_loss: 4.7139
```

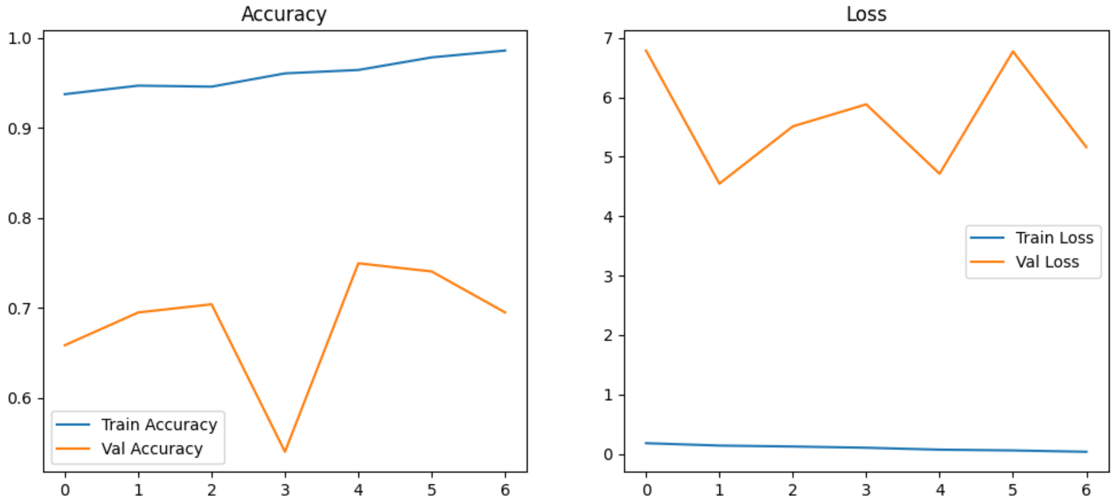
```
Epoch 6/30
83/83              5s 54ms/step -
accuracy: 0.9713 - loss: 0.0754 - val_accuracy: 0.7405 - val_loss: 6.7726
Epoch 7/30
83/83              5s 56ms/step -
accuracy: 0.9853 - loss: 0.0372 - val_accuracy: 0.6950 - val_loss: 5.1609
```

[12]:

```
# Plot Accuracy & Loss
def plot_history(history):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.legend()
    plt.title('Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.legend()
    plt.title('Loss')
    plt.show()

plot_history(history)
```



[13]:

```
# Evaluate Model
loss, accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
```

```
21/21              1s 44ms/step -
```

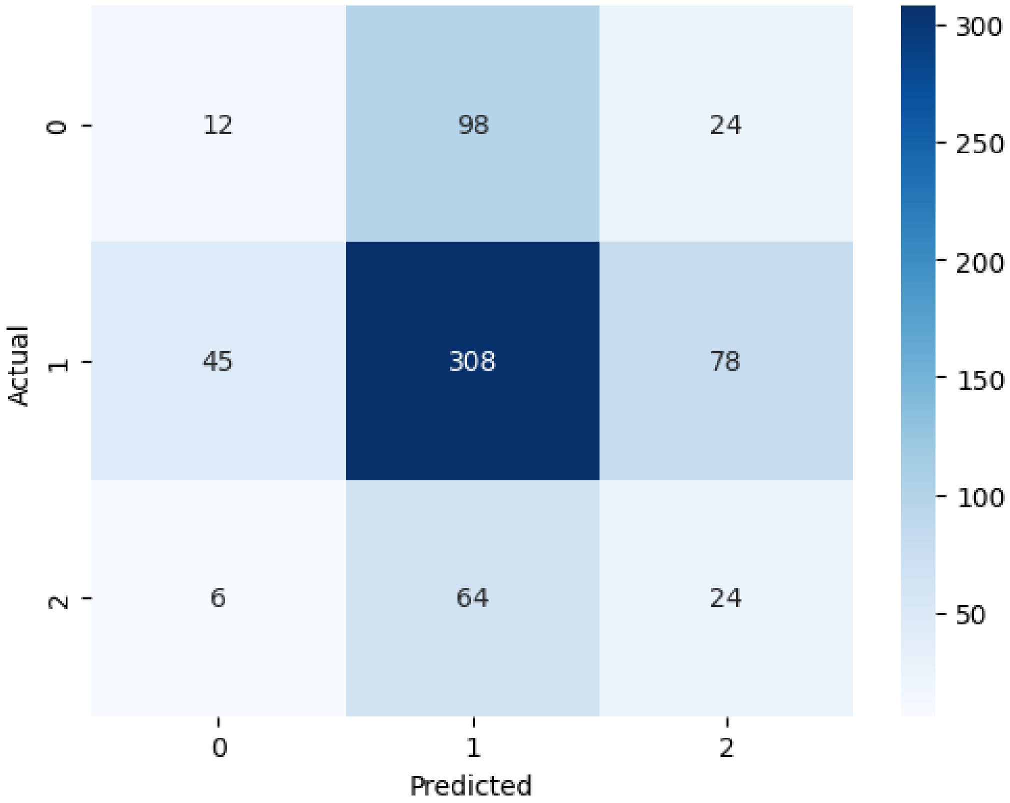
```
accuracy: 0.6846 - loss: 6.9851
Validation Accuracy: 69.50%
```

[14]:

```
# Confusion Matrix
y_true = val_generator.classes
y_pred = model.predict(val_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
cm = confusion_matrix(y_true, y_pred_classes)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
21/21            1s 56ms/step
```



[15]:

```
# Classification Report
print(classification_report(y_true, y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.19      0.09      0.12       134
           1       0.66      0.71      0.68       431
           2       0.19      0.26      0.22        94

    accuracy                           0.52       659
   macro avg       0.35      0.35      0.34       659
weighted avg       0.49      0.52      0.50       659
```

[16]:

```python
# Predict on New Image
def predict_new_image(image_path, model):
    img = cv2.imread(image_path)
    img = cv2.resize(img, IMG_SIZE)
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    pred = model.predict(img)
    class_idx = np.argmax(pred)
    class_label = list(train_generator.class_indices.keys())[class_idx]
    print(f"Predicted Class: {class_label}")

predict_new_image("/kaggle/input/capsule-endoscopy-dataset-kauhc/Normal/
 ↪Normal_2024-08-07-07-47-35_10102.bmp", model)
```

```
1/1               1s 602ms/step
Predicted Class: AVM
```