Automated Teeth Disease Detection Using Deep Learning

Ahmed Mostafa Gamal Eddin
Computer Vision intern
Cellula Technologies

July 2, 2025

Contents

1 Introduction

Teeth diseases can lead to severe health problems if left undiagnosed. However, manual classification by professionals is time-consuming and error-prone. With the rise of deep learning, especially Convolutional Neural Networks (CNNs), automated image classification can aid medical professionals in faster diagnosis.

2 Problem Statement

The goal of this project is to develop a CNN model that can accurately classify teeth diseases from images into one of seven categories:

- OC (Oral Cancer)
- CaS (Caries)
- OT (Other Types)
- CoS (Cosmetic Issues)
- Gum (Gum Disease)
- MC (Molar Conditions)
- OLP (Oral Lichen Planus)

3 Dataset Description

The dataset is divided into three subsets:

1. **Training:** Used to train the model
2. **Validation:** Used to tune hyperparameters and monitor performance
3. **Testing:** Final evaluation

Dataset Hierarchy

```
Teeth_Dataset/
 Training/
 Validation/
 Testing/
```

4 procedures and steps

- Load and Explore the data by showing images for each class
- Apply data augmentation to enhance generalization.
- Apply data augmentation to enhance generalization.
- train CNN model to process the data
- evaluate the model using various metrics like accuarcy classification report .

5 Load and explore the data

The dataset is organized into three main directories: Training, Validation, and Testing, each containing subfolders named after specific disease classes. These subfolders contain the corresponding images. The loading process reads these directories and automatically assigns labels based on folder names, ensuring that the data is structured appropriately for supervised learning.

Code Snippet

```
base_path = "/kaggle/input/teeth-dataset/Teeth_Dataset"

train_data = tf.keras.utils.image_dataset_from_directory(
    base_path + "/Training",
    image_size=(224, 224),
    batch_size=32
)

val_data = tf.keras.utils.image_dataset_from_directory(
    base_path + "/Validation",
    image_size=(224, 224),
    batch_size=32
)

test_data = tf.keras.utils.image_dataset_from_directory(
    base_path + "/Testing",
    image_size=(224, 224),
    batch_size=32,
    shuffle=False
)
)
```

Listing 1: Load the data

6 plot the distribution of the data

Figure 1: Distribution of dataset

7 Plotting some photos for each class

Code Snippet

```
Illness = ['OC', 'CaS', 'OT', 'CoS', 'Gum', 'MC', 'OLP']
Types = ['Validation', 'Training', 'Testing']
def plot_one_image_per_illness(category, illness_list):

    fig, axes = plt.subplots(1, len(illness_list), figsize=(10, 10))
    if len(illness_list) == 1:
        axes = [axes]

    for ax, illness in zip(axes, illness_list):
        # illness_dir = "/kaggle/input/teeth-dataset/Teeth_Dataset/OC"
        illness_dir = os.path.join(base_path, category, illness)
        valid_extensions = ('.jpeg', '.jpg', '.bmp', '.png')
        #illness_image = list of the path of each image
        illness_images = list(filter(lambda x: x.endswith(valid_extensions
    ), os.listdir(illness_dir)))

        if illness_images:
            img_path = os.path.join(illness_dir, random.choice(
    illness_images))
            img = plt.imread(img_path)
            ax.imshow(img)
            ax.set_title(illness)
            ax.axis('off')
        else:
            ax.set_title(f'No images for {illness}')
```

```
            ax.axis('off')

    plt.tight_layout()
    plt.show()


plot_one_image_per_illness(Types[0], Illness)
plt.suptitle('Random Images [Validation Dataset]:', y=1.00)

plot_one_image_per_illness(Types[1], Illness)
plt.suptitle('Random Images [Training Dataset]:', y=1.00)

plot_one_image_per_illness(Types[2], Illness)
plt.suptitle('Random Images [Test Dataset]:', y=1.00)
```

Listing 2: Load the data

7.1 some random images

Figure 2: Distribution of dataset

8 Preprocessing and Augmentation

The `Load_and_preprocessing()` function is responsible for preparing the training and validation datasets before feeding them into the Convolutional Neural Network (CNN). It performs two essential tasks:

8.0.1 Image Augmentation (Training Set Only)

To improve the generalization of the model and prevent overfitting, the training images undergo a series of random transformations:

- **Rescaling:** Pixel values are normalized from the range $[0, 255]$ to $[0, 1]$.

- **Data Augmentation Techniques:**
  - Random rotation
  - Width and height shifts
  - Shearing transformation
  - Zooming in and out
  - Horizontal flipping
  - Filling of missing pixels using the nearest strategy

8.0.2 Image Normalization (Validation Set)

For the validation set, only rescaling is applied:

- **Rescaling:** Pixel values are normalized to the $[0, 1]$ range.
- **No augmentation:** Validation images remain unchanged to ensure an accurate evaluation of model performance.

8.0.3 Data Generator

Both training and validation sets are passed through the `flow_from_directory()` method which:

- Automatically assigns labels based on folder names.
- Resizes images to the specified `target_size` (e.g., 150x150).
- Returns batches of image-label pairs using one-hot encoding.

8.1 Code Snippet

```
def Load_and_preprocessing(train_data, validation_data, target_size,
    batch_size):
    train_datagen = ImageDataGenerator(
        rescale=1.0/255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range = 0.2,
        shear_range= 0.2,
        zoom_range=0.2,
         horizontal_flip=True,
        fill_mode='nearest'
    )

    validation_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_data,
```

```
        target_size = (150,150),
        batch_size=batch_size,
        class_mode = 'categorical'
    )

    validation_generator = validation_datagen.flow_from_directory(
        validation_data,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='categorical'
    )


    return train_generator, validation_generator
```

Listing 3: Image Preprocessing and Augmentation

8.2 Augmented vs original

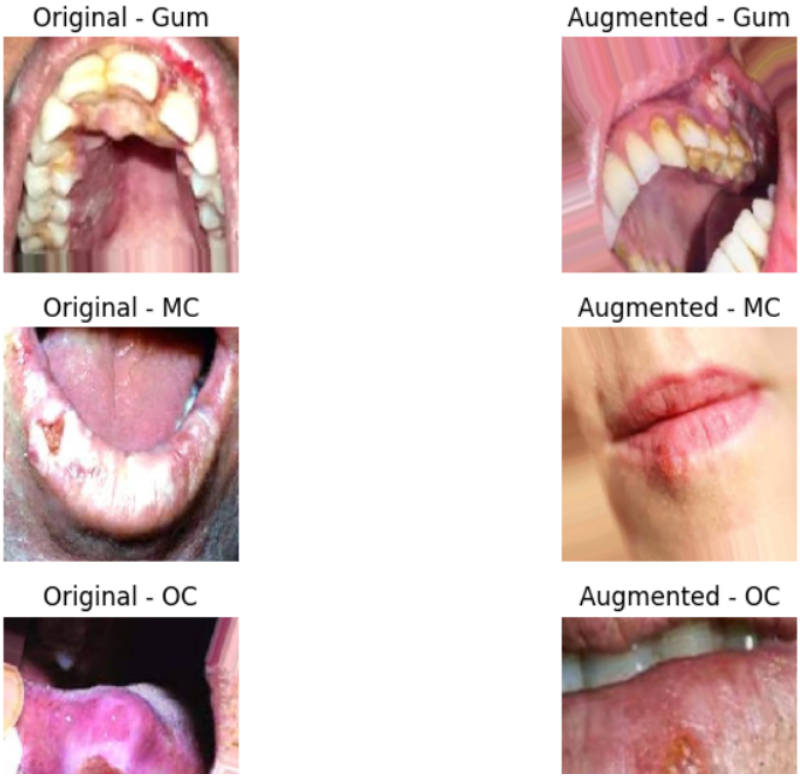

Figure 3: Augmented vs original

∈

9 CNN Model Architecture

The Convolutional Neural Network (CNN) model used in this project is designed to classify input dental images into 7 disease categories. It follows a sequential architecture consisting of multiple convolutional and pooling layers, followed by dense layers for classification.

Layer-by-Layer Breakdown

- **Conv2D (32 filters, 3×3 kernel, ReLU activation):**
  Extracts 32 local features from the input image using small 3×3 filters with non-linearity introduced via ReLU.
- **MaxPooling2D (2×2 pool size):**
  Downsamples the feature maps by taking the maximum value in each 2×2 region, reducing dimensionality and computation.
- **Conv2D (64 filters, 3×3 kernel, ReLU activation):**
  Increases the depth of the network to learn more complex patterns and features.
- **MaxPooling2D (2×2 pool size):**
  Further reduces the spatial dimensions to retain only the most prominent features.
- **Conv2D (128 filters, 3×3 kernel, ReLU activation):**
  Adds more abstraction by learning high-level features, useful for distinguishing fine-grained disease types.
- **MaxPooling2D (2×2 pool size):**
  Reduces spatial dimensions again to prepare for the dense layers.
- **Flatten:**
  Converts the 2D output of the last pooling layer into a 1D vector for input into the fully connected layers.
- **Dense (512 units, ReLU activation):**
  A fully connected layer acting as the decision-making layer with ReLU activation to introduce non-linearity.
- **Dropout (rate = 0.5):**
  Randomly drops 50% of neurons during training to prevent overfitting and improve generalization.
- **Dense (7 units, Softmax activation):**
  Final output layer that produces a probability distribution over the 7 disease classes for classification.

Code Snippet

```
model = Sequential([
    Conv2D(32,(3,3),activation='relu', input_shape=(150,150,3)),
    MaxPooling2D((2,2)),
    Conv2D(64,(3,3),activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(128,(3,3),activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(7, activation='softmax')
])
```

Listing 4: CNN Architecture

10 Training Configuration

Loss Function: Categorical Crossentropy
Optimizer: Adam
Metrics: Accuracy
Callbacks: EarlyStopping and ReduceLROnPlateau

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
early_stopping = EarlyStopping(monitor='val_loss', patience=4,
    restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
    min_lr=0.00001)
```

Listing 5: Callbacks for Training

11 train and Evaluation

After 100 epochs the model achieved the following:

- Training Accuracy: accuracy: 0.9112 - loss: 0.2765
- Validation Accuracy: accuracy: 0.9786 - loss: 0.0863
- Test Accuracy: accuracy: 0.9838 - loss: 0.0679
- Classification Report and Confusion Matrix generated using `sklearn.metrics`

Visualizations

Figure 4: Training and Validation accuracy over Epochs

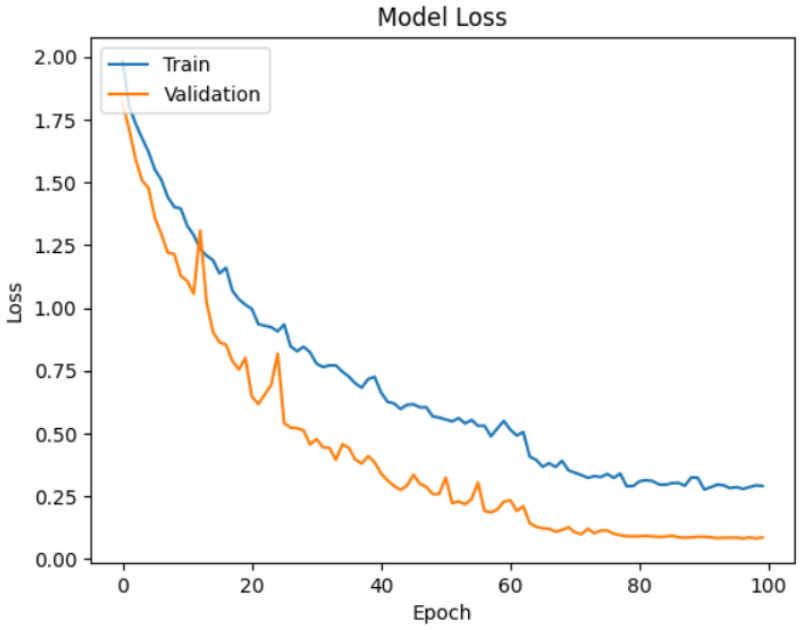

Figure 5: Training and Validation loss over Epochs

Classification Report

	precision	recall	f1-score	support
CaS	0.99	0.99	0.99	160
CoS	1.00	0.99	0.99	149
Gum	0.99	0.98	0.99	120
MC	0.97	0.96	0.97	180
OC	0.95	0.98	0.96	108
OLP	0.98	0.98	0.98	180
OT	0.98	0.99	0.98	131
accuracy			0.98	1028
macro avg	0.98	0.98	0.98	1028
weighted avg	0.98	0.98	0.98	1028

Figure 6: classification report

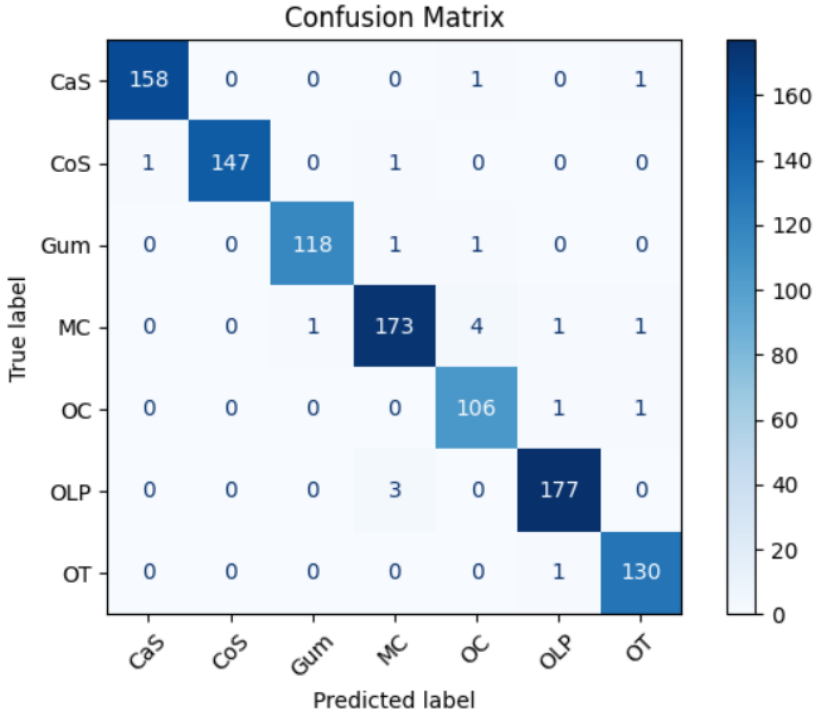

Figure 7: confusion matrix

12 Conclusion

In this project, we built a deep learning pipeline for classifying teeth diseases with decent accuracy. The CNN model learned to differentiate between 7 types of diseases using real-world image data. With further tuning, it could support clinical diagnosis as a decision support tool.