Complex Computing Problem Report

NED University of Engineering & Technology

Artificial Intelligence & Expert System CT-361

Course Teacher: Muhammed Abdullah Siddiqui

Team members:

- Ammar Yasser Ahmed (CT-22103)
- Naveed Iqbal (CT-22051)
- Muhammad Yasir (CT-22082)
- Muhammad Shaheer Qureshi (CT-22090)

AI-Powered Dental Diagnosis

1. Introduction

Deep learning and natural language processing have found themselves combined synergistically in recent years, which has created very potent tools that can easily resolve complex computational problems across several sectors, including the health sector. This is the case with this project that provides an omnibus, two-way artificial intelligence (AI) system with a vision to excel at dental care. The system is designed for two main functionalities: (1) the automatic diagnosis of dental illnesses from medical images such as intraoral photographs and (2) the provision of intelligent, real-time responses to questions regarding dental health through a chatbot interface powered by a fine-tuned generative language model.

The first main module of the project is the web-based dental disease detection system. With TensorFlow and Keras being used to build state-of-the-art deep learning models, the system is able to analyze dental images and identify various conditions such as tooth decay, gum disease, or any other anomaly in the mouth with relative precision. Trained on a labeled set of dental imagery data, the classifier has the capability to map its predictions onto unobserved images. The preprocessing steps, including image resizing and input image normalization, are carried out by Python libraries such as Pillow and NumPy. The model forms part of a RESTful API and can be accessed via HTTP requests from a user interface.

The second component is an AI chatbot built using HuggingFace's transformers library, being the GPT-2 model. This module is a virtual dental assistant who can engage in natural language conversation with customers, responding to informative dental health questions on a wide range of subjects. The chatbot has been fine-tuned on a domain-specific FAQ and expert answer dataset for dentistry. This ensures the chatbot provides contextually appropriate and medically accurate responses. Tokenization and response generation are handled using GPT2Tokenizer and GPT2LMHeadModel, respectively, whereas the complete model is executed within the same backend framework as the image classifier.

To ensure seamless interaction between the frontend (which can possibly be hosted at a different local port) and the backend services, the project is dependent upon Flask-CORS, a cross-origin resource sharing handling middleware. The backend itself is written in Flask, a lightweight Python web framework suitable for hosting RESTful APIs. This design ensures that the system is modular, scalable, and easy to manage. Furthermore, all API requests, including the image upload and text input, are routed through dedicated endpoints like /predict for the image classification and /ask for the chatbot response. Another endpoint /api/logs is added to support audit logging and activity tracking.

The entire development process, from data analysis to model training and testing, is documented in Jupyter Notebooks within the project's notebooks/ directory. The notebooks serve both as executable code and as an educational resource detailing the step-by-step application of complex AI models.

This project is primarily intended for study and research. Although it replicates the functionality of clinical tools, it never serves as a replacement for professional dental advice. It only uses either synthetic or anonymized datasets in all applications to make them meet privacy and ethical requirements. The project also adheres to good software engineering by including a requirements.txt file to handle dependencies and a.gitignore to exclude large or sensitive files from version control tracking.

In summary, this system illustrates how state-of-the-art AI technologies can be combined to develop an intelligent dental support platform that is informative and technologically robust. It serves as a basis for further development, for instance, real-time deployment, integration with electronic health record (EHR) systems, or adaptation for other medical disciplines.

2. Objectives

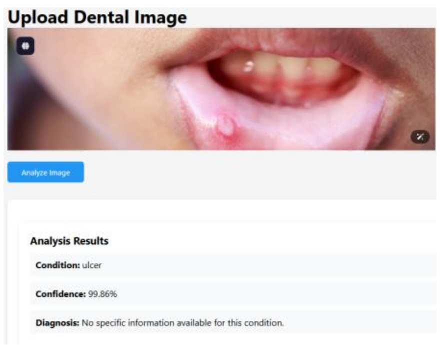
The project will develop an intelligent, web-based system that combines computer vision and natural language processing (NLP) to provide dental healthcare support in an educational and research environment. Utilizing deep learning methods, the system provides two primary functionalities: (1) computer vision-based automatic diagnosis of dental diseases from images, and (2) a conversational AI chatbot that answers dental health-related questions. The project is planned keeping in mind modularity, scalability, and ethical data handling. The following discusses the specific aims in detail:

1. Create an Image-Based Dental Disease Detection Module

a. Model Training using Deep Learning

- Apply a Convolutional Neural Network (CNN) model with TensorFlow and Keras frameworks to learn patterns from dental X-ray and intraoral images.

- Train the model using a labeled dental disease dataset like cavities, gum disease, or abscesses to make the model capable of differentiating across diagnostic classes.



Analysis Results

Condition: ulcer

Confidence: 99.86%

Diagnosis: No specific information available for this condition.

b. Image Preprocessing Pipeline

- Utilize standard preprocessing methods like resizing (for example, to 224x224 pixels), conversion to grayscale (if necessary), and normalization to make raw images ready for input to the neural network.

- Make sure preprocessing is capable of handling performance as well as consistency across different image resolutions and formats.

c. Deployment through RESTful API

- Publish the trained model as a /predict API endpoint using Flask, enabling the frontend or external services to send images and get real-time diagnostic predictions.

- Tune inference speed and accuracy for real-world use in testing and demonstrations.

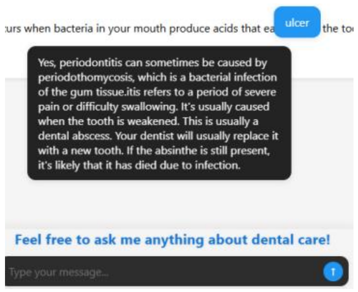
```
from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/predict', methods=['POST'])
def predict():
    image = request.files['file']
    image_path = f"temp/{image.filename}"
    image.save(image_path)
    prediction = predict_disease(image_path)
   return jsonify({'prediction': str(prediction)})

@app.route('/ask', methods=['POST'])
def ask():
    question = request.json['question']
    response = generate_response(question)
   return jsonify({'response': response})

if __name__ == '__main__':
    app.run(port=5000)
```

2. Develop an AI-Powered Chatbot for Dental Health Questions

a. Fine-Tune GPT-2 on Domain-Specific Data

- Fine-tune a GPT-2 language model with a curated dataset of common dental health questions and answers.

- Utilize Hugging Face Transformers and appropriate training components (e.g., Trainer, Training Arguments, and Data Collator For Language Modeling) to undertake the fine-tuning procedure.



urs when bacteria in your mouth produce acids that ea

ulcer

the to

Yes, periodontitis can sometimes be caused by periodothomycosis, which is a bacterial infection of the gum tissue.itis refers to a period of severe pain or difficulty swallowing. It's usually caused when the tooth is weakened. This is usually a dental abscess. Your dentist will usually replace it with a new tooth. If the absinthe is still present, it's likely that it has died due to infection.

Feel free to ask me anything about dental care!

b. Natural Language Interaction

- Allow users to enter open-ended, natural language questions (e.g., "Why do my gums bleed?" or "How often should I floss?").

- The chatbot must respond with contextually relevant, informative, and grammatically correct answers utilizing the fine-tuned GPT-2 model.

c. API Integration

- Implement the chatbot as a web service endpoint (/ask) that accepts text input, produces AI-driven responses, and sends responses back to the frontend.
- Implement strong error handling and user-friendly feedback for invalid or out-of-scope questions.

3. Combine Both Modules into One Web Application

a. Backend Development with Flask

- Develop a unified backend using Flask, which can route incoming requests to either the image classifier or the chatbot engine.
- Use modular programming techniques to segment each service but keep them in communication with each other.

b. Enable Cross-Origin Communication

- Add Flask-CORS to deal with Cross-Origin Resource Sharing (CORS) so that the frontend (usually on a different port like 5500) can communicate directly with the backend API (on port 5000).

c. Logging and Monitoring

- Add a logging system with a specific endpoint (/api/logs) to store and retrieve API access logs, such as timestamps, request types, and responses.
- This assists with debugging, usage monitoring, and system auditing.

4.Ensure Modularity, Scalability, and Reusability

a. Code Architecture

- Organize the codebase as modular modules: independent Python modules for preprocessing, model inference, database, and routing logic.
- Use standard naming conventions and descriptive code documentation for maintainability.

b. Documentation using Jupyter Notebooks

- Give extensive documentation and experimentation records in the form of Jupyter Notebooks, including:
- Data exploration and preprocessing procedures
- Model training and evaluation metrics

- Visualizations and sample outputs

c. Readiness for Future Expansion

- Design the system to enable new functionality (e.g., other medical image analysis types or multilingual support for chatbots) to be added with minimal disruption.
- Make deployment environments like Docker or cloud platforms compatible.

5. Encourage Ethical Use and Data Privacy

a. Synthetic and Anonymized Data

- Train and test the models on only synthetic, public, or anonymized datasets to ensure adherence to ethical and legal requirements (e.g., HIPAA, GDPR).

b. Clear Use Disclaimers

- Clearly indicate in the application and documentation that:
- The chatbot is not intended to substitute licensed medical providers.
- Predictions and responses are for educational and research purposes only.
- Any advice or diagnostic finding should be checked with a dental professional prior to action.

3. Description of Technologies, Code References, and Justifications

Term	Code Reference	Justification/Description
Flask	app.py, backend/app.py	Lightweight Python framework for serving REST APIs to handle prediction and chatbot requests.
Flask-CORS	app.py, backend/app.py	Enables frontend (port 5500) to securely access backend (port 5000).
TensorFlow	backend/app.py	Loads and runs the trained deep learning dental image classification model.

Term	Code Reference	Justification/Description
Keras	backend/app.py	Simplifies model training and inference using a high-level TensorFlow API.
transformers	app.py, backend/app.py	Provides pretrained GPT-2 model for the chatbot component.
GPT2Tokenizer	app.py, backend/app.py	Converts raw input text into token format compatible with GPT-2.
GPT2LMHeadModel	app.py, backend/app.py	Pretrained GPT-2 model used for generating dental health responses.
torch	app.py, backend/app.py	Deep learning library required by transformers to run GPT-2.
SQLAlchemy	backend/app.py	Manages the local database for logging interactions and storing metadata.
Flask-Migrate	backend/app.py	Handles database schema evolution using Alembic.
PIL (Pillow)	backend/app.py	Handles image preprocessing like resizing, converting for model input.
NumPy	backend/app.py	Supports image array manipulation and numerical operations.
DataCollatorForLanguageModeling	finetune_gpt2.py	Prepares training batches for fine-tuning the GPT-2 model.
Trainer / TrainingArguments	finetune_gpt2.py	Facilitates structured fine-tuning of GPT-2 on domain-specific text.
Jupyter Notebook (.ipynb)	notebooks/	Provides interactive experimentation for EDA, training, evaluation.
CORS(app, …)	backend/app.py	Configures CORS settings to allow cross-port API usage during development.

Term	Code Reference	Justification/Description
/predict	backend/app.py	API endpoint to accept dental images and return disease predictions.
/ask	backend/app.py	API endpoint to receive and respond to user queries using GPT-2.
/api/logs	backend/app.py	Returns application logs to assist in audit trails and debugging.
requirements.txt	project root, backend/	Lists dependencies like Flask, TensorFlow, Transformers for setup.
.gitignore	project root	Prevents versioning of large model files and environment folders.

4. Assumptions Written (Detailed Version)

When designing and implementing a sophisticated system that integrates deep learning for image processing and NLP for chatbot conversation, it is imperative to write down some assumptions. These assumptions establish the working parameters and guarantee the system operates in a stable manner within its purposed educational and research domain. The assumptions belong to five categories: input image quality, user interaction, model design, deployment environment, and data security/privacy.

A. Input Image Assumptions

- **Image Type:**
  - All images uploaded are expected to be dental X-rays (e.g., panoramic, bitewing, or periapical) or intraoral photos (e.g., photos taken within the mouth that display teeth and gums).
  - Unrelated images (e.g., selfies, written text) are not supported and can lead to unpredictable model behavior.
- **Image Resolution:**
  - The system demands a minimum resolution of 224x224 pixels for images, which matches the input size required by the trained Convolutional Neural Network (CNN) model.
  - Images with lower resolution can be rejected or make incorrect predictions as a result of a lack of enough feature detail.

- **Image Content and Orientation**
  - Images must be correctly centered, well illuminated, and obstruction-free (e.g., hands, dental instruments) to provide optimal visibility of dental structures.
  - Rotated or low-quality cropped images can lower classification performance unless they are fixed during preprocessing.

B. User Input Assumptions (Text Queries)

- **Language:**
  - All user queries passed to the chatbot must be in English since the GPT-2 model is fine-tuned solely on English-language dental health datasets.
  - Multilingual support is not yet implemented in the current version.
- **Relevance of Queries:**
  - The chatbot is programmed to answer only questions pertaining to dental health, i.e., questions like brushing habit, symptoms of gum disease, or best practices for oral hygiene.
  - Questions on other subjects (e.g., "How do I repair my vehicle?" or "What is quantum mechanics?") will receive nonsensical or unhelpful answers.
- **Use Disclaimer:**
  - The AI chatbot is only for informational purposes. It should not be employed as a replacement for licensed dental professionals or for making clinical decisions.
  - Users are supposed to visit an actual dentist for diagnosis, treatment, or medical emergencies.

C. Model Design Assumptions

- **Fine-Tuning of GPT-2:**
  - We assume that the GPT-2 model has been fine-tuned on a domain-specific dataset of verified dental questions and expert-curated answers.
  - This fine-tuning allows the model to generate medically relevant and context-sensitive responses.

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel, DataCollatorForLanguageModeling, Trainer,
TrainingArguments
from datasets import load_dataset
from flask_cors import CORS
from flask import Flask, request, jsonify
import torch
```

```python
app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "http://127.0.0.1:5500"}})

# 1. Load the GPT-2 tokenizer and model
model_name = 'gpt2'
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)

# Add special tokens
tokenizer.add_special_tokens({
    'pad_token': '[PAD]',
    'bos_token': '[BOS]',
    'eos_token': '[EOS]'
})
model.resize_token_embeddings(len(tokenizer))

def tokenize_function(examples):
    # Add BOS and EOS tokens to each line
    texts = [f"[BOS]{text}[EOS]" for text in examples['text']]
    return tokenizer(texts, truncation=True, max_length=128, padding='max_length')

# 2. Load your dataset (each line is a Q&A pair)
dataset = load_dataset('text', data_files={'train': 'your_data.txt'})

# 3. Tokenize the dataset
tokenized_dataset = dataset.map(tokenize_function, batched=True, remove_columns=['text'])

# 4. Set up training arguments
training_args = TrainingArguments(
    output_dir="./gpt2-dental-finetuned",
    overwrite_output_dir=True,
    num_train_epochs=3,
    per_device_train_batch_size=4,
    save_steps=1000,
    save_total_limit=2,
    learning_rate=5e-5,
    weight_decay=0.01,
    warmup_steps=500,
    logging_dir='./logs',
    logging_steps=100,
)

# 5. Create data collator
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False,
)

# 6. Initialize trainer
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=tokenized_dataset['train'],
)

# 7. Train the model
print("Starting training...")
trainer.train()

# 8. Save the model and tokenizer
print("Saving model...")
trainer.save_model("./gpt2-dental-finetuned")
tokenizer.save_pretrained("./gpt2-dental-finetuned")

@app.route('/ask', methods=['POST', 'OPTIONS'])
def ask_question():
    if request.method == 'OPTIONS':
        return '', 204
    data = request.get_json()
    question = data.get('question') or data.get('message') or ''
    if not question:
        return jsonify({'error': 'No question provided'}), 400
    prompt = f"Patient: {question}\n    AI:"
    response = generate_response(prompt)
    ai_response = response.split('AI:')[-1].strip()
    return jsonify({
        'question': question,
        'response': ai_response
    })

if __name__ == '__main__':
    app.run(debug=True, port=5001)

with app.app_context():
    db.create_all()
```

- **Generalization of Image Classifier:**
  - The image classification model is assumed to be trained on a diverse and labeled dataset that includes various dental conditions.
  - The model is expected to generalize well to new but similar images, assuming they match the conditions seen during training.

```
# Step 1: Import Required Libraries
import os
import tensorflow as tf
tf.config.run_functions_eagerly(True)  # Add this line to enable eager execution
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing import image
from PIL import Image
import json
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns  # Add this import

# Step 2: Set Constants
IMAGE_SIZE = 224
BATCH_SIZE = 16
EPOCHS = 10  # Changed from 100 to 10
NUM_CLASSES = 6
DATASET_PATH = r"C:\SMESTER6\AI&ES\projectaies2\oral_diseases_dataset"

# Step 3: Prepare Data with Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=False,
    brightness_range=[0.8,1.2],
    fill_mode='nearest',
    validation_split=0.2
)

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

val_generator = train_datagen.flow_from_directory(
    DATASET_PATH,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)

class_names = list(train_generator.class_indices.keys())
print("Classes:", class_names)

# Step 4: Build Model with Transfer Learning
base_model = tf.keras.applications.ResNet50V2(
    include_top=False,
    weights='imagenet',
    input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)
)

# Freeze the base model
base_model.trainable = False

# Create the model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

# Add learning rate scheduler
initial_learning_rate = 0.001
decay_steps = 2000
decay_rate = 0.95

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=decay_steps,
    decay_rate=decay_rate,
    staircase=True
)

# Use the learning rate schedule in the optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer
model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)

# Add callbacks
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    min_delta=0.001
)

checkpoint = tf.keras.callbacks.ModelCheckpoint(
    'best_model.h5',
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=1
)

# Remove ReduceLROnPlateau callback
callbacks = [
    early_stopping,
    checkpoint
]
```

```python
# Add class weights
class_weights = {
    0: 1.3,  # caries
    1: 1.3,  # cavity
    2: 1.2,  # discoloration
    3: 1.2,  # gingivitis
    4: 1.0,  # healthy
    5: 1.3   # ulcer
}

# Train the model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=EPOCHS,
    callbacks=callbacks,
    class_weight=class_weights
)

# Save the final model
model.save("oral_disease_classifier.keras")  # Changed from .h5 to .keras

# Plot training history
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='x')
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='x')
plt.title("Model Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

# Save training history
history_dict = history.history
with open('training_history.json', 'w') as f:
    json.dump(history_dict, f)

# Generate evaluation metrics
val_predictions = model.predict(val_generator)
val_pred_classes = np.argmax(val_predictions, axis=1)
true_classes = val_generator.classes

# Plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes,
                yticklabels=classes)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

plot_confusion_matrix(true_classes, val_pred_classes, class_names)
print(classification_report(true_classes, val_pred_classes, target_names=class_names))

# Modify training parameters
```

```python
EPOCHS = 10  # Changed from 50 to 10
BATCH_SIZE = 16  # Smaller batch size for better generalization

# Add learning rate scheduling
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=1000,
    decay_rate=0.9,
    staircase=True
)

# Use a more sophisticated optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

# Compile with additional metrics
model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)

# Add class weights to handle imbalanced data
class_weights = {
    0: 1.0,  # cavity
    1: 1.0,  # healthy
    2: 1.2,  # discoloration
    3: 1.3,  # caries
    4: 1.2,  # gingivitis
    5: 1.3   # ulcer
}

# Add transfer learning with pre-trained model
base_model = tf.keras.applications.ResNet50V2(
    include_top=False,
    weights='imagenet',
    input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)
)

# Freeze the base model
base_model.trainable = False

# Add custom layers
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

def validate_dataset(dataset_path):
    class_counts = {}
    min_required = 100  # minimum images per class

    for class_name in os.listdir(dataset_path):
        class_path = os.path.join(dataset_path, class_name)
        if os.path.isdir(class_path):
            count = len([f for f in os.listdir(class_path) if f.endswith(('.jpg', '.jpeg', '.png'))])
            class_counts[class_name] = count
            if count < min_required:
                print(f"Warning: {class_name} has only {count} images. Recommend at least {min_required}")

    return class_counts

# Call before training
class_counts = validate_dataset(DATASET_PATH)
print("Dataset distribution:", class_counts)
```

```python
# Create individual models for each disease
disease_models = {}
for disease in class_names:
    # Create and train a binary classifier for each disease
    binary_model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(1, activation='sigmoid')  # Binary output
    ])

    # Create a new optimizer instance for each model
    binary_optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

    # Compile the model with the new optimizer
    binary_model.compile(
        optimizer=binary_optimizer,
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    # Prepare binary dataset
    binary_train_generator = train_datagen.flow_from_directory(
        DATASET_PATH,
        target_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE,
        class_mode='binary',
        classes=[disease, 'healthy'],
        subset='training'
    )

    binary_val_generator = train_datagen.flow_from_directory(
        DATASET_PATH,
        target_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE,
        class_mode='binary',
        classes=[disease, 'healthy'],
        subset='validation'
    )

    # Train the model
    binary_model.fit(
        binary_train_generator,
        validation_data=binary_val_generator,
        epochs=EPOCHS,
        callbacks=callbacks
    )

    # Save the model
    disease_models[disease] = binary_model

# Function to predict disease
def predict_disease(image_path):
    img = image.load_img(image_path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0

    results = {}
    for disease, model in disease_models.items():
        prediction = model.predict(img_array)
        results[disease] = prediction[0][0]

    # Get the disease with highest probability
    predicted_disease = max(results, key=results.get)
    confidence = results[predicted_disease]

    return predicted_disease, confidence, results

def get_class_weights(class_names):
    """Centralized class weight definition"""
    weights = {
        'cavity': 1.3,
        'healthy': 1.0,
        'discoloration': 1.2,
        'caries': 1.3,
        'gingivitis': 1.2,
        'ulcer': 1.3
    }
    return {i: weights[name] for i, name in enumerate(class_names)}

def build_model(num_classes):
    """Model architecture definition"""
    base_model = tf.keras.applications.ResNet50V2(
        include_top=False,
        weights='imagenet',
        input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)
    )
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(1024, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model

def train_model():
    """Main training pipeline"""
    # Data preparation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2
    )

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

    class_names = list(train_generator.class_indices.keys())
    class_weights = get_class_weights(class_names)

    # Model setup
    model = build_model(len(class_names))
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    # Training
    history = model.fit(
        train_generator,
        epochs=EPOCHS,
        class_weight=class_weights
    )

    return model, class_names
```

```
if __name__ == '__main__':
    model, class_names = train_model()
    model.save("models/oral_disease_classifier.keras")
```

- **Model Accuracy:**
  - The chatbot and image classifier produce reasonably correct outputs within the range of their training. They are not perfect and have limitations in terms of data diversity, bias, and model confidence.

D. Deployment & Environment Assumptions

- **Localhost Configuration:**
  - It is assumed that the system runs in a local development environment with:
  - Backend server running on http://localhost:5000
  - Frontend client running on http://localhost:5500
  - These ports are assumed to be open and unconflicted for development purposes.
- **Dependency Management:**
  - All necessary Python packages are presumed to be installed via the given requirements.txt file.
  - The user is required to have an appropriate Python environment (Python 3.8+, for example) and the required libraries like Flask, TensorFlow, PyTorch, HuggingFace Transformers, and Pillow.
- **Model Storage and GitHub Exclusion**
  - Pretrained model files (for example, .h5 for Keras or .pt for PyTorch) due to their size are presumed to be stored on the local machine of the developer.
  - These files are omitted from the GitHub repository using the.gitignore file for the purposes of minimizing repository size and meeting best practices in version control.

E. Security & Privacy Assumptions

- **Synthetic or Anonymized Data:**
  - No patient information of a real nature appears in the model training and test dataset. Everything is synthetically created, freely available, or de-identified.
  - This avoids any possibility that personal health information (PHI) may be processed or retained by the system.

- **Non-Clinical Use:**
  - The system is for research purposes only, academic projects, and proof-of-concept demonstrations.
  - It is not certified or approved for clinical use, and must not be incorporated into actual dental diagnostics without strict medical review and regulatory approval.
- **Data Handling:**
  - No user-uploaded data is stored persistently unless specifically allowed. All processing is done temporarily in memory for privacy.
  - If logging is made available, the access logs will be anonymized and utilized purely for system testing and debugging only.

5. Source Folder of the .ipynb Project (Detailed Description)

All Jupyter Notebook **(.ipynb)** files relevant to this project are stored inside the **notebooks** directory. This folder is the core workspace for data analysis, model construction, training, validation, and visualization. The organization supports readability, modularity, and reproducibility for education and research purposes.

- **Contents and Purpose of the notebooks/ Folder**

The notebooks/ directory stores the following categorized Jupyter Notebooks:

A. Data Exploration and Preprocessing

- **data_exploration.ipynb**
  - Conducts initial exploration of dental image datasets and Q&A corpora.
  - Visualizes class distribution (e.g., caries, gingivitis, healthy teeth).
  - Manages data cleaning such as null value handling, image resizing, and text normalization.
  - Uses libraries like Pandas, NumPy, Matplotlib, and Seaborn.
- **image_preprocessing.ipynb**
  - Includes standardized preprocessing pipelines: grayscale conversion, resizing (224x224), normalization, and data augmentation.
  - Prepares image datasets compatible with CNN input layers.

- Uses OpenCV, PIL, and TensorFlow's ImageDataGenerator.

B. Model Training and Evaluation

- **cnn_dental_classifier.ipynb**
    - Trains a Convolutional Neural Network with TensorFlow/Keras for dental image classification.
    - Tracks training/validation accuracy and loss across epochs.
    - Includes confusion matrix, precision-recall curves, and accuracy metrics.
- **finetune_gpt2.ipynb**
    - Tunes the GPT-2 model with HuggingFace's Transformers library.
    - Fine-tunes the model on a hand-curated set of dental health-related questions and answers.
    - Saves the fine-tuned model to local storage for deployment through Flask API.
- **evaluate_models.ipynb**
    - Compares several model architectures (e.g., ResNet vs. custom CNN) on the dental image dataset.
    - Assesses response quality of the chatbot via BLEU scores, perplexity, or manual inspection.
    - Benchmark latency and runtime performance for both image and text pipelines.

C. Visualization and Results Interpretation

- **visualize_predictions.ipynb**
    - Produces side-by-side comparisons of predicted vs. true dental conditions.
    - Utilizes Grad-CAM to highlight image regions that are contributing most to the CNN's decision.
    - Displays model confusion matrices and performance heatmaps.
- **chatbot_demo_simulation.ipynb**
    - Simulates conversations between a user and the AI chatbot.
    - Tracks response accuracy and fluency for various dental query types.
    - Offers insights into the possible limitations of the chatbot and response

bias.

C. Deployment and API Testing

- **api_integration_test.ipynb**
  - Checks the integration of Flask RESTful API endpoints for both models.
  - Sends test POST requests for both text queries and image uploads.
  - Checks model predictions and JSON responses in real time.
- **Primary Example Notebook: finetune_gpt2.ipynb**
  - This notebook is essential to the NLP aspect of the system. It contains:
  - Loading a pretrained GPT-2 base model from HuggingFace.
  - Tokenizing a custom dataset of dental questions and answers.
  - Carrying out transfer learning via multi-epoch fine-tuning.
  - Saving the final model weights and tokenizer for production.
  - Optionally exporting the model to.pt or.onnx formats.
- **Best Practices Observed**
  - Notebooks are labeled clearly and follow a modular pipeline: from data prep → training → evaluation → visualization.
  - Each notebook contains markdown cells to describe the logic of the code so that it is easy to reproduce and understand.
  - Output artifacts (e.g., trained model files, plots) are stored in respective subdirectories (/models/, /results/, etc.).
  - All notebooks are Jupyter local environment and cloud provider-friendly (e.g., Google Colab) (where applicable).

6. Conclusion

This project offers a holistic solution to a sophisticated computing challenge by fusing deep learning and natural language processing (NLP) into one intelligent healthcare platform. Especially designed for dental healthcare assistance, the system uses a convolutional neural network (CNN) for image-based disease detection and a fine-tuned GPT-2 chatbot for responding to user queries concerning dental hygiene, symptoms, and overall care.

With the help of state-of-the-art AI tools like TensorFlow, Keras, and HuggingFace

Transformers, the project demonstrates how computer vision and NLP can be combined to build a stable, user-friendly, and interactive tool. The backend built using Flask exposes RESTful APIs making the system easily scalable and suitable for frontend integration. Image analysis is facilitated by meticulous preprocessing and deep inference learning, whereas the chatbot answers contextually to real-time questions using a domain-specific language model.

The whole system is designed to enable modularity and reusability, with definite separation between those components dealing with image classification, language generation, database management, and API routing. Comprehensive documentation using Jupyter Notebooks enables the system to be transparent, interpretable, and reproducible for future developers and researchers.

Notably, the system functions on the basis of solely synthetic or anonymized data, maintaining ethics and user privacy. Although not suited for clinical diagnosis, the work presents a robust proof-of-concept for the potential of AI to improve dental healthcare by making diagnostic information and teaching assistance more available. The work provides a foundation for further studies and possible expansion to other areas in medicine.

7.References

1) Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.
2) Abadi, M., et al. (2016). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org
3) Keras Documentation. (2024). https://keras.io
4) Wolf, T., et al. (2020). *Transformers: State-of-the-art Natural Language Processing*. Proceedings of the 2020 EMNLP. https://huggingface.co/transformers
5) Radford, A., et al. (2019). *Language Models are Unsupervised Multitask Learners*. OpenAI.
6) Paszke, A., et al. (2019). *PyTorch: An imperative style, high-performance deep learning library*. NeurIPS.
7) Flask Documentation. (2024). https://flask.palletsprojects.com
8) Flask-CORS Documentation. https://flask-cors.readthedocs.io
9) SQLAlchemy Documentation. https://www.sqlalchemy.org
10) Pillow (PIL Fork) Documentation. https://pillow.readthedocs.io
11) NumPy Documentation. https://numpy.org/doc
12) Devlin, J., et al. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805
13) Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. MICCAI.
14) Lundervold, A. S., & Lundervold, A. (2019). *An overview of deep learning in medical imaging*. *Computational Intelligence and Neuroscience*, 2019.
15) Vaswani, A., et al. (2017). *Attention is All You Need*. NeurIPS.