Transfer Learning-based Classification of Poultry Diseases for Enhanced Health Management

Introduction:

1. Shaik Shavul Hameed(Gathering the complete project related Information)
2. Joshva siril(Installation of the softwares related to front- End and Back-End)
3. kesav reddy(Importing the libraries and Building the Model)
4. Manjunadh(Running & Evaluating the Model)
5. imroz(Visualizing & Manual Testing)

Transfer Learning-based Classification

of PoultryDiseases for Enhanced Health Management

Project Overview:

This project focuses on developing an intelligent system for the classification of poultry diseases. By employing **transfer learning** with pre-trained deep learning models, the goal is to significantly improve the accuracy and speed of early disease detection in poultry, which is crucial for effective health management and maintaining farm productivity. The system primarily involves analyzing images (often of fecal samples) to distinguish between healthy and diseased conditions, and potentially to categorize specific disease types.

Key Features:

- **Transfer Learning**:

Leverages the power of established Convolutional Neural Networks (CNNs) like VGG16, ResNet, MobileNetV3Small, or DenseNet as foundational models, pre-trained on vast image datasets. This allows the model to learn complex features from a wide range of images, which are then adapted to the specific task of poultry disease classification.

- **Automated Disease Classification**:

The core functionality revolves around classifying images to identify various poultry diseases (e.g., Coccidiosis, Salmonella, Newcastle disease).

- **Modular Architecture**:

The codebase is structured in a modular fashion, breaking down the machine learning pipeline into distinct, manageable stages. This enhances code readability, maintainability, and facilitates independent testing of components.

- **Data Version Control (DVC)**:

Implements DVC for robust data and pipeline management. This ensures

reproducibility of experiments, tracking of dataset versions, and efficient collaboration.

- **Interactive Web Application**:
Features a user-friendly web interface developed with Flask, enabling easy image uploads for disease prediction.

- **Cloud Deployment Ready**:
Designed with considerations for Continuous Integration/Continuous Deployment (CI/CD), making it suitable for deployment on cloud platforms like AWS or Azure.

2.Technologies Used:

The project is built upon a foundation of popular and powerful tools and libraries:

- **Python**: The primary programming language (commonly Python 3.8 or higher is recommended).
- **TensorFlow/Keras**: The leading deep learning framework for model development, training, and evaluation.
- **Pre-trained CNN Models**: VGG16, ResNet, MobileNetV3Small, DenseNet – serving as feature extractors in the transfer learning approach.
- **Flask**: A lightweight Python web framework used for developing the front-end user interface.
- **Flask-Cors**: A Flask extension to handle Cross-Origin Resource Sharing (CORS) for web requests
- **DVC (Data Version Control)**: Essential for managing datasets, models, and ML pipelines, ensuring reproducibility and collaborative development.
- **NumPy**: Fundamental library for numerical computations
- **Pillow (PIL)**: For image loading and basic manipulation.
- **Scikit-learn**: Utilized for various machine learning utilities, including data splitting and performance metric calculations.
- **Matplotlib**: For data visualization and plotting graphs (e.g., training history, confusion matrices).
- **Jupyter Notebook**: Used for exploratory data analysis, prototyping, and step-by-step development in the `research` directory.
- **YAML**: For structuring configuration parameters in files like `config.yaml` and `params.yaml`.

3. Project Workflow:

- **Data Ingestion (`01_data_ingestion`):**
  Purpose: Acquiring and preparing the raw dataset.
  Process: This stage involves downloading, extracting, and performing initial organization of the poultry disease image dataset. Images are categorized (e.g., "Disease" or "Healthy," or specific disease types).

- **Prepare Base Model (`02_prepare_base_model`)**:
  Purpose: Setting up the pre-trained CNN for transfer learning.
  Process: A pre-trained CNN (e.g., VGG16) is loaded. Its final classification layers are removed, as these are specific to the original training task. The model is configured with an input shape that matches the new poultry image dataset, and the base model's weights are typically frozen to retain learned features. The prepared base model is then saved.

- **Training (`03_training`)**:
  Purpose: Adapting and training the model on the poultry disease dataset.
  Process: The prepared base model is loaded, and new custom classification layers (e.g., `GlobalAveragePooling2D` followed by `Dense` layers) are added on top. The model is then compiled with an appropriate optimizer (e.g., Adam), a loss function (e.g., `binary_crossentropy` for two classes), and relevant metrics (e.g., accuracy). The model is trained on the prepared dataset, often incorporating data augmentation techniques to improve robustness. The final trained model is saved (e.g., as `model.h5`).

- **Evaluation (`04_evaluation`)**:
  Purpose: Assessing the trained model's performance.
  Process: The trained model is loaded and evaluated against an independent test dataset. Key performance metrics such as accuracy, loss, precision, recall, and F1-score are calculated. Detailed reports like classification reports and confusion matrices are generated to provide insights into the model's strengths and weaknesses.

- **Prediction Pipeline**:
  Purpose: Enabling the model to make predictions on new, unseen images.
  Process: A dedicated pipeline is set up to load the final trained model. It handles the necessary preprocessing steps for any input image before feeding it to the model for classification.

- **User Application Integration (`app.py`)**:
  Purpose: Providing an accessible interface for users to interact with the model.
  Process: A Flask web application (`app.py`) is developed. This application manages image uploads from users, sends them through the prediction pipeline, and displays the classification results in a user-friendly manner, typically using HTML templates (e.g., `index.html`).

4. Setup and Local Environment Configuration:

To get your project up and running on a local machine, follow these steps:
Prerequisites:

(Anaconda/Miniconda): Highly recommended for creating and managing isolated Python environments.
Git: For cloning the project repository from GitHub.
Installation Steps:
Clone the Project Repository: Open your terminal or command prompt and execute:

Bash:

- git clone https://github.com/Shavulhameed16/Transfer-Learning-based-Classification-of-Poultry-Diseases-for-Enhanced-Health-Management.git
- cd Transfer-Learning-based-Classification-of-Poultry-Diseases-for-Enhanced-Health-Management.

Create and Activate Conda Environment: It's best practice to create a dedicated environment to avoid conflicts with other Python projects.
Bash:

- conda create -n poultry_disease_env python=3.8 -y # You can name the environment differently and choose a compatible Python version
- conda activate poultry_disease_env

Install Required Dependencies: The project's dependencies are listed in requirements.txt. Install them using pip:
Bash:
pip install -r requirements.txt

(If there's also a setup.py and you want to install in editable mode for development, you might see pip install -e . in some instructions, which ensures your local changes are reflected.)

Initialize DVC (if the project uses it extensively): If your project leverages DVC for data and pipeline management, initialize it and pull any versioned data:
Bash:
dvc init
dvc pull

Configure Project Parameters:

Review and potentially adjust settings in the config/config.yaml and params.yaml files. These files control data paths, model hyperparameters, training configurations, and other crucial settings.

Running the Project:

1. **Execute the Machine Learning Pipeline (Optional)**: If you wish to re-run the entire data processing, training, and evaluation pipeline, use DVC's reproducibility command:

Bash:

```
dvc repro
```

Alternatively, you can run individual stages, starting from the Jupyter notebooks in the research folder for step-by-step execution

- **Start the Prediction Web Application:** To launch the user-facing web application, navigate to the project's root directory and run:

```
python app.py
```

The application will typically become accessible in your web browser at

http://127.0.0.1:5000 (localhost on port 5000).

Project Structure (Typical Layout):

A standard organization for a project like this helps in navigation and understanding:

```
├── artifacts/                  # Stores generated files (e.g., datasets, base models, trained models)
|   ├── data_ingestion/
|   ├── prepare_base_model/
```

```
|    └── training/
├── config/                       # Configuration files
|    └── config.yaml
├── params.yaml                   # Parameters for model training,
DVC stages
├── dvc.yaml                      # DVC pipeline definition
├── research/                     # Jupyter notebooks for
experimentation and initial development
|    ├── 01_data_ingestion.ipynb
|    ├── 02_prepare_base_model.ipynb
|    └── ...
├── src/                          # Source code for modularized
components
|    └── cnnClassifier/          # Main package for the project
|        ├── __init__.py
|        ├── components/         # Individual pipeline components
|        ├── config/
|        ├── entity/
|        ├── pipeline/           # ML pipeline stages (data
ingestion, training, etc.)
|        ├── utils/
|        └── ...
├── app.py                        # Flask web application entry point
├── main.py                       # Main script for running the DVC
pipeline
├── templates/                    # HTML templates for the web app
(e.g., index.html)
├── Dockerfile                    # For containerization
├── .dvcignore                    # Files/folders to be ignored by
DVC
├── .gitignore                    # Files/folders to be ignored by
Git
├── requirements.txt              # Python dependencies
├── setup.py                      # Python package setup
└── README.md                     # This project's main documentation
```

EXPECTED output:

Actual output: