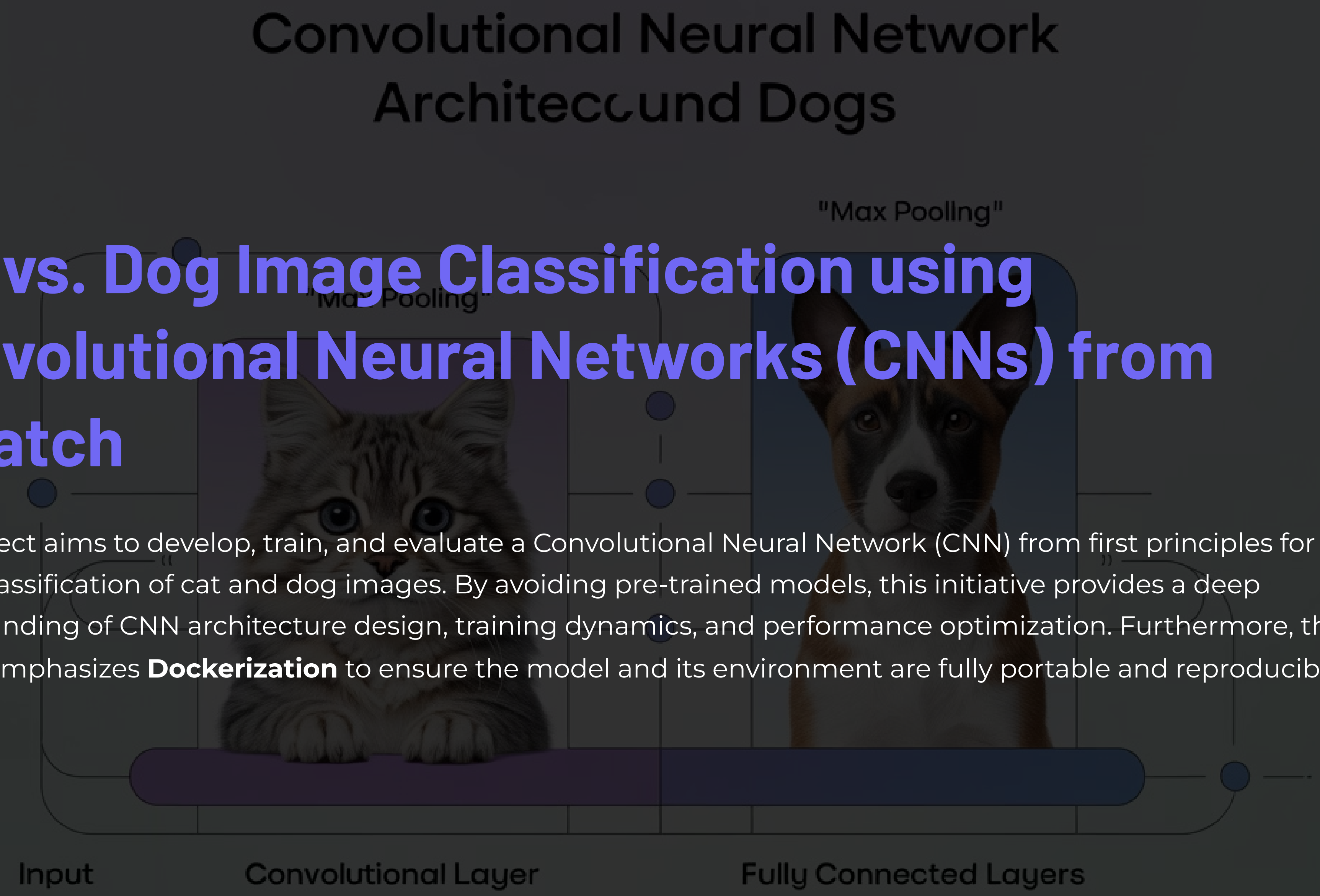

Cat vs. Dog Image Classification using Convolutional Neural Networks (CNNs) from Scratch

This project aims to develop, train, and evaluate a Convolutional Neural Network (CNN) from first principles for the binary classification of cat and dog images. By avoiding pre-trained models, this initiative provides a deep understanding of CNN architecture design, training dynamics, and performance optimization. Furthermore, the project emphasizes **Dockerization** to ensure the model and its environment are fully portable and reproducible.

Problem Statement

The challenge involves constructing an accurate image classification model from the ground up to effectively distinguish between images of cats and dogs. This necessitates the careful design of a CNN architecture and meticulous handling of the dataset to ensure the model generalizes robustly to unseen data without relying on transfer learning. A key aspect will also be packaging the entire solution using Docker for consistent execution across different environments.

Learning Outcomes

CNN Fundamentals

Comprehend and implement the core principles of CNNs, including convolutional layers, pooling, activation functions, and dropout.

Image Preprocessing & Augmentation

Gain practical experience in preparing image datasets, applying preprocessing techniques, and utilizing data augmentation to enhance model robustness and prevent overfitting.

Model Lifecycle Management

Develop proficiency in the end-to-end process of training, validating, and testing deep learning models.

Performance Evaluation

Evaluate model performance using a comprehensive suite of metrics, including accuracy, confusion matrices, precision, recall, and F1-score.

Overfitting Mitigation

Implement strategies such as dropout and data augmentation to effectively reduce overfitting in deep learning models.

Model Persistence

Learn to save and load trained models for deployment and future use.

Containerization with Docker

Understand and implement Docker to containerize the CNN training and inference environment, ensuring reproducibility and portability.

Dataset

The project utilizes the widely recognized **Kaggle Cat vs. Dog Dataset**, comprising thousands of labeled images of cats and dogs.

- **Dataset Link:** https://www.kaggle.com/datasets/karakaggle/kaggle-cat-vs-dog-dataset
- **Structure:** Images are logically organized into Cat/ and Dog/ subdirectories.
- **Preprocessing:** Images will be loaded and preprocessed using Keras's ImageDataGenerator for essential operations such as rescaling, resizing, and on-the-fly data augmentation.
- **Validation Split:** A common practice of an 80% training and 20% validation split will be employed to monitor model performance during training and detect overfitting.

Data Preparation & CNN Architecture

Data Preparation

- **Loading & Preprocessing:** Leverage ImageDataGenerator for efficient loading and initial image preprocessing (e.g., pixel value rescaling).
- **Data Augmentation:** Implement real-time data augmentation techniques (e.g., rotation, horizontal flips, width/height shifts) to artificially expand the dataset and improve model generalization.
- **Splitting:** Partition the dataset into training and validation sets using an 80/20 ratio.

CNN Architecture Design

A custom CNN model will be constructed from scratch, incorporating the following essential layers:

- **Input Layer:** Accepts image input, resized to a consistent dimension (e.g., $150 \times 150 \times 3$ for color images).
- **Convolutional Layers (Conv2D):** Multiple layers with Rectified Linear Unit (ReLU) activation functions to extract hierarchical features from the input images.
- **Pooling Layers (MaxPooling2D):** Used to downsample feature maps, reducing spatial dimensions and computational complexity while retaining salient features.
- **Dropout Layers:** Strategically placed layers to prevent overfitting by randomly setting a fraction of input units to zero at each update during training, forcing the network to learn more robust features.
- **Flatten Layer:** Converts the 2D feature maps output from convolutional layers into a 1D vector, preparing them for the fully connected layers.
- **Fully Connected (Dense) Layers:** One or more dense layers with ReLU activation to learn complex non-linear relationships from the flattened features.
- **Output Layer:** A single dense layer with a sigmoid activation function for binary classification, outputting a probability between 0 and 1.

Model Training & Evaluation

Model Compilation

- **Loss Function:** Binary Cross-Entropy, suitable for binary classification tasks.
- **Optimizer:** Adam optimizer, known for its adaptive learning rate capabilities and efficiency.
- **Metrics:** Accuracy will be tracked as the primary performance metric during training and evaluation.

Training

The model will be trained for a suitable number of epochs (e.g., 10-20), with careful consideration for batch size (e.g., 32) based on available hardware resources. Training and validation accuracy/loss will be closely monitored to identify signs of overfitting.

Evaluation

- **Performance Curves:** Plots of training and validation loss/accuracy across epochs will be generated to visualize learning progress and detect overfitting.
- **Confusion Matrix:** A confusion matrix will be computed on the validation/test dataset to provide a detailed breakdown of correct and incorrect classifications.
- **Classification Metrics:** Precision, recall, and F1-score will be calculated to offer a comprehensive assessment of the classifier's quality, particularly valuable in imbalanced datasets (though less critical here).

Experiments

- **Architectural Tuning:** Modify the number of convolutional layers, the number of filters per layer, and kernel sizes.
- **Dropout Rate Optimization:** Experiment with different dropout rates to find the optimal balance between regularization and information retention.
- **Data Augmentation Parameters:** Adjust the parameters of data augmentation techniques (e.g., rotation range, shift ranges) to assess their impact on model robustness.

Outputs & Deliverables

Project Outputs

- **Trained Model:** The final trained CNN model saved in the HDF5 format (.h5).
- **Performance Plots:** Visualizations of training and validation accuracy and loss over epochs.
- **Evaluation Summary:** A summary of the confusion matrix and calculated classification metrics.
- **Docker Assets: A Dockerfile and associated files for building the project's Docker image.**

Complete Deliverables

- **Codebase:** A well-commented and organized Jupyter Notebook or Python script containing all project code.
- **Visualizations:** High-quality plots showcasing training/validation accuracy and loss, and the confusion matrix.
- **Experimental Summary:** A concise summary of experimental results and insights gained from architectural and hyperparameter tuning.
- **Final Model File:** The saved trained model file, named cat_dog_cnn_from_scratch.h5.
- **Docker Assets:** A comprehensive Dockerfile and any necessary configuration files to build and run the project within a Docker container.

Extensions & Tools

Optional Extensions

- **Early Stopping:** Implement early stopping callbacks during training to automatically halt training when validation performance plateaus, preventing overfitting.
- **Learning Rate Scheduling:** Employ learning rate scheduling techniques to dynamically adjust the learning rate during training, potentially leading to faster convergence and improved performance.
- **Model Deployment:** Develop a simple web or mobile interface to deploy the trained model for interactive image classification.
- **Docker Compose: Utilize Docker Compose for orchestrating multi-container applications, if the project grows to include separate services (e.g., a web UI and a model serving API).**

Tools & Libraries

Python

The foundational programming language for the project.

TensorFlow / Keras

The primary framework for building, training, and evaluating CNN models.

Matplotlib / Seaborn

For generating informative data visualizations and plots.

scikit-learn

For computing and presenting various classification evaluation metrics.

Docker

For containerization, ensuring consistent and reproducible environments for development, training, and deployment.