**Image-Based Entity Prediction for Unit and Value Models**

**Objective:**

The aim is to develop a machine learning system that predicts both numerical values and corresponding units (such as weight, length, or volume) for different entities from product images. This process involves training two models: one for predicting numerical values and another for predicting the units.

**Process Overview:**

1. **Dataset Preparation and Image Downloading:**
   * The dataset contains images and entity-related values (such as weight, voltage, etc.).
   * Images are downloaded based on links provided in the dataset. Placeholder images are created if downloading fails.
   * The dataset is processed to clean and normalize the entity values and their units (like handling ranges or plural forms of units).
2. **Preprocessing:**
   * Values and units are extracted and cleaned from the dataset.
   * Images are preprocessed and resized to a fixed size (128x128 pixels) to ensure uniform input to the models.
   * The system checks for invalid units or values, skipping rows with errors.
3. **Training Two Models:**
   * **Numerical Value Prediction Model**: This is a regression model that predicts numerical values for the entity (e.g., the weight of an object in kilograms).
   * **Unit Prediction Model**: This is a classification model that predicts the corresponding unit (e.g., gram, kilogram, or pound) for the entity.
4. **Model Architecture:**
   * **Numerical Value Model**: A deep convolutional neural network (CNN) with multiple Conv2D layers, BatchNormalization, MaxPooling, and Dropout layers to avoid overfitting. It outputs a single value.
   * **Unit Prediction Model**: Another CNN but designed as a classifier, predicting the correct unit from a predefined list of units.
5. **Data Preprocessing for Models:**
   * The images are loaded and normalized to a [0, 1] range.
   * For the value prediction model, numerical labels (entity values) are standardized using StandardScaler.
   * For the unit prediction model, units are label-encoded into integer values, which are used for classification.
6. **Training Procedure:**
   * Both models are trained using the dataset. The training data is split into training and validation sets (80/20).
   * Early stopping is applied to prevent overfitting, and checkpoints are used to save the best model during training.
7. **Predictions on Test Data:**
   * After training, the models can predict numerical values and units for unseen test images.
   * The output for each image is a predicted value along with its corresponding unit.
8. **Sanity Check:**
   * A separate Python script (sanity.py) is used to ensure that the generated predictions pass certain checks.

**Code Sections**

1. **Imports and Constants:** Libraries like tensorflow, pandas, and multiprocessing are used. Key constants such as IMAGE\_SIZE, BATCH\_SIZE, and EPOCHS are defined to control model parameters.
2. **Download Image Functionality:**
   * The function download\_image() handles downloading each image from the given URL.
   * A placeholder image is generated if an image download fails.
3. **Unit and Value Normalization:**
   * Various helper functions (common\_mistake(), clean\_entity\_value(), normalize\_unit()) handle the parsing of the entity values and units.
   * These functions take into account errors like different forms of units (e.g., "cm" vs "centimetre") and incorrect formats like ranges.
4. **Data Preprocessing and Loading:**
   * The load\_and\_preprocess\_data() function handles loading the dataset, downloading images, cleaning entity values, and preparing the data for model training.
   * Images are resized, entity values are cleaned, and units are encoded.
5. **Model Building:**
   * The function build\_value\_model() creates a CNN for predicting numerical values.
   * The function build\_unit\_model() builds another CNN for unit classification.
6. **Training the Model:**
   * The function train\_model() handles training both models using the dataset.
   * It also splits the data into training and validation sets, applies early stopping, and saves the best models to disk.
7. **Prediction on Test Images:**
   * The predict\_entity\_value() function takes a test image and predicts both the entity's value and its unit using the trained models.

**Key Functions:**

1. **download\_image()**:
   * Downloads the image from a given link or generates a placeholder if the download fails.
2. **normalize\_unit()**:
   * Maps variations of units (e.g., "litre" and "liter") to a standard form.
3. **clean\_entity\_value()**:
   * Cleans and extracts numerical values from strings, handling ranges or lists of values.
4. **build\_value\_model()**:
   * Constructs a CNN to predict the entity's numerical value.
5. **build\_unit\_model()**:
   * Constructs a CNN to classify the unit (e.g., kg, lb) from a predefined list of units.

**Execution:**

1. **Training:**
   * Run the train\_model() function, which loads the data, downloads the images, and trains the models.
2. **Prediction:**
   * For test images, the function predict\_entity\_value() can be used to generate predictions.
3. **Sanity Check:**
   * A final sanity check is done using sanity.py to verify the output predictions.

**Improvements:**

* **Unit Normalization**: The system normalizes units and maps common mistakes to correct forms (e.g., "liters" -> "litres").
* **Error Handling**: Skips invalid rows or rows with missing images and handles errors during image downloading.
* **Model Regularization**: Dropout layers are added to the models to prevent overfitting.

**Conclusion:**

The system effectively processes images to predict both the numerical value and the corresponding unit for an entity, providing a robust solution for analyzing visual data related to measurements in various units.