Model Architecture:

CNN-based Feature Extraction Model:
A **Convolutional Neural Network** (CNN) model can be used to extract visual features from images. Here is a breakdown of the architecture:

Input Layer:
Images: Input size will be the standardized image size (e.g., 224x224 or 299x299 pixels).

Convolutional Layers:
Apply multiple convolutional layers to detect features like edges, textures, and product boundaries. Use **ReLU** activation after each convolutional layer.

Max-Pooling Layers:
Max-pooling layers are used to reduce the **spatial dimensions** of the feature maps, maintaining important information while reducing computational cost.

Dense Layers:
Flatten the output from the convolutional layers.
Fully connected (dense) layers with dropout to prevent overfitting.

Entity Prediction Network:
Once the CNN has extracted image features, a separate branch can predict the entity value and unit.

Entity Value Regression:
The **extracted features** are passed to a fully connected network to predict the entity value (e.g., weight, volume). This is a **regression** task.

Unit Classification:
The same extracted features are passed through another branch of the network, which performs a **classification task** to predict the correct unit (e.g., gram, liter, volt).

Loss Functions:
Mean Squared Error (MSE) for predicting the entity value.
Cross-Entropy Loss for unit classification.

Combined Loss:
A weighted combination of the two loss functions to ensure the model optimizes for both the entity value and unit simultaneously.

Training Pipeline:

1: **Preprocessing**:
Download images and store them locally.
Preprocess images for input into the CNN.
Preprocess entity_name and entity_value columns in the training set.

Step 2: **Model Training**:
Use ImageDataGenerator to load and augment training data.
Train the CNN model using both image data and categorical information (such as group_id).
Optimize the model using Adam optimizer with a learning rate schedule.

Step 3: **Validation**:
Use a portion of the training data as a validation set to monitor model performance and avoid overfitting.

Step 4: **Post-Processing**:
Convert predicted entity values into the required float and unit format.
Ensure output predictions are formatted exactly as per the given constraints.

Evaluation:

The model will be evaluated based on the **F1** score, using the following steps:
1.Compare predicted values (OUT) with the ground truth (GT).
2.Ensure predictions adhere to the correct unit standards.
3.The F1 score will be calculated based on precision and recall of the predictions.

Output Formatting & Sanity Check:

Ensure all predictions are formatted according to the rules outlined in the problem statement.
Use sanity.py to check if the final CSV passes all formatting checks before submission.