Amazon ML Challenge Report

**Team Name - RCB**

**Problem Statement**- Feature Extraction from Images

In this hackathon, the goal is to create a machine learning model that extracts entity values from images. This capability is crucial in fields like healthcare, e­ commerce, and content moderation, where precise product information is vital. As digital marketplaces expand, many products lack detailed textual descriptions, making it essential to obtain key details directly from images. These images provide important information such as weight, volume, voltage, wattage, dimensions, and many more, which are critical for digital stores

**Machine Learning Approach and Model Overview**

The provided code outlines a comprehensive workflow for a machine learning (ML) project focused on image data partnered with additional features (in this case, entity names and values). Below is a detailed report that includes the approach, the models used, experiments conducted, and conclusions drawn from the analysis.

**1. Objective and Dataset Overview**

The primary objective of this project is to predict a continuous numeric output (entity values) from images. The dataset comprises images linked to certain entities, which are annotated with labels (entity names) and their corresponding values. The goal is to build a predictive model that can efficiently learn from both the image data and these features.

**2. Data Preparation**

The workflow begins by importing necessary libraries, mounting Google Drive, and loading data files including training and test datasets using Pandas’ read\_csv function. The dataset is then sampled down to 1% to facilitate faster experimentation and reduce computational load.

**Image Downloading:**

The code includes a function to download images from URLs specified in the dataset to a target directory. In cases where the image fails to download, appropriate error handling is implemented to ensure robustness.

**Unit Conversion:**

A crucial preprocessing step involves converting various entity values (like weights and volumes) into a standard numerical format. The \_convert\_to\_numeric function is defined for this purpose and handles multiple units, ensuring that the model can learn from these values uniformly.

**3. Data Pipeline and Transformations**

To prepare the data for use in a neural network, the project defines a custom ImageDataset class that extends PyTorch’s Dataset class. This class loads images and applies necessary preprocessing transformations like resizing, normalization, and tensor conversion to images.

**Data Splitting:**

The dataset is split into training and testing subsets using the train\_test\_split function from Scikit-learn. This ensures that model performance can be evaluated objectively on unseen data.

**4. Model Architecture**

The code implements a Convolutional Neural Network (CNN) using the ResNet-18 architecture pre-trained on ImageNet. This model is designed to extract image features effectively before passing these features to a fully connected layer for generating predictions.

**Model Definition:**

* Feature Extraction: The CNN processes the images through several layers, capturing essential characteristics.
* Prediction Layer: The final fully connected layer outputs a predicted value based on the extracted features.

This architecture facilitates learning from both visual features and possibly numeric encodings of the entity names (though in the provided code, the entity names are not directly used in the computation).

**5. Training Procedure**

The model is trained over a defined number of epochs (10 epochs in this case) using the Mean Squared Error (MSE) loss function because the target variable is continuous. An Adam optimizer is employed for training, providing an adaptive learning rate.

Training Loop:

* The model enters a training mode, processes the training data, and computes the loss.
* The loss is backpropagated to update the model’s weights.
* The average loss for each epoch is printed, allowing monitoring of training progress.

**6. Testing Procedure**

The testing phase evaluates the model's performance on unseen data, using a similar loop to the training phase but without weight updates. The total loss on the test set is calculated to gauge how well the model generalizes beyond the training data.

**7. Results and Conclusion**

While the code does not produce detailed metrics or visualizations, the loss values printed during training and testing provide a basic understanding of model performance.

1. **Model Performance**: By analyzing the training and test losses, one can infer the model's learning ability. A significant difference (high training loss vs. low test loss) may indicate overfitting, while similar values suggest effective generalization.
2. **Data Augmentation**: Future work can involve data augmentation techniques (like flipping, rotation, etc.) to increase model robustness and performance.
3. **Hyperparameter Tuning**: Exploring different hyperparameters (learning rate, batch size, number of epochs) could potentially improve the results.
4. **Integration of Entity Names**: The code mentions an optional pathway for incorporating the entity name features into the model, which deserves deeper examination for improving predictive accuracy.
5. **Conclusion**: The project demonstrates a structured approach to building an image-based predictive model leveraging deep learning techniques. By utilizing transfer learning through a pre-trained ResNet model, it aims to efficiently capture image features relevant for predicting numeric outputs associated with those images. Further refinements and expansions can enhance its performance and adaptability.

This report outlines the basic workflow of the code while offering insights into potential directions for future work, focusing on both the technical and conceptual aspects of the ML model being developed.