Meesho Visual Taxonomy Data Challenge

Team ML Maverick - Private Leaderboard Rank 23 (score: 0.7693)

Members: Neel Shah (neeldevenshah.ai@gmail.com), Sneh Shah (22aiml049@charusat.edu.in), Harsh Maheshwari (22aiml019@charusat.edu.in)

1. Introduction

Brief Summary: Our approach used a deep learning solution that leveraged a single model architecture with attribute-specific classifier heads to predict the attributes for various categories of clothing items. The base model, facebook/convnext-base-384-22k-1k, was employed as a feature extractor, with a shared feature layer followed by distinct classifier heads for each attribute. Initial training was conducted on the entire model, followed by fine-tuning the classifier heads for individual attributes to enhance performance. This unified model structure, combined with strategic training, enabled accurate and efficient attribute prediction across categories.

2. Data Preprocessing

Data Cleaning:
The initial dataset contained significant missing values. To address this, we first dropped all rows with missing values, resulting in a data reduction from 18,346 to approximately 2,500 rows for one category (e.g., Sarees). This reduction led to data redundancy and poor representation of minor classes. To overcome this limitation, we employed a two-phase training strategy to maximize data utilization and enhance model performance:

1. **Phase One – Full Model Training**: The entire model was initially trained using a combination of Binary Cross-Entropy (BCE) loss and Maximum Mean Discrepancy (MMD) loss. This phase involved training the pre-trained base model's shared features and all attribute classifier heads simultaneously. This ensured that the model learned generalizable features and relationships across the dataset, helping mitigate the effects of missing data and domain variation.
2. **Phase Two – Attribute-Specific Fine-Tuning**: After the initial training, if an attribute (e.g., Attribute 1) had missing data (e.g., 10,000 rows), we froze all layers except the classifier head specific to that attribute. We then fine-tuned the specific attribute classifier head using all available data for that attribute, applying only BCE loss. This phase ensured that the model specialized in learning the most from the available data for each attribute, leading to improved handling of class imbalance and minor class representation.

Note: The numbers provided (e.g., 18,346 to 2,500 rows) are specific to one category (Sarees) and serve as an illustrative example for the dataset.

Data Exploration and Learnings:
The data exploration revealed severe class imbalance, especially after the removal of missing values. Attributes such as color, sleeve length, and type had varying distributions across different clothing categories.
Note: The class distribution and imbalance are observed differently for each category and should be considered as an example for one category.

Training and Validation Data Split Strategy:
A **90-10 train-test split** was used for model training and evaluation, applied to each category separately.

Feature Engineering:
Image augmentations were applied to improve the model's robustness. Each training image had a **50% probability of being augmented**, and augmentations varied with each epoch to enhance generalization.
Image size was set to **512x512 pixels** for improved performance, as larger input sizes yielded better results compared to the base model's original **338x338** training size.

3. Modeling Approach

Model Selection:

- The base model chosen was facebook/convnext-base-384-22k-1k, pre-trained on 22k ImageNet data and fine-tuned on 1k ImageNet data, for robust feature extraction.
- Attribute-specific models were built on top of the shared feature base, with separate classifiers (convolutional and dense layers) for each attribute.

Architecture:

- The base model and extracted shared features from images.
- Classifier layers for each attribute comprised convolutional layers followed by dense layers, tailored to the number of classes for each attribute.

Base Model	ConvNeXt-Base pre-trained on ImageNet-22k and fine-tuned on ImageNet-1k(`facebook/convnext-base-384-22k-1k`).
Input Image Size	512 - 512 pixels (larger than original 384- 384 for enhanced detail).
Backbone Features	Output from the ConvNeXt backbone with hidden size of 1024 (for the base model)
Feature Processing Layer	Convolutional layer (1 × 1 kernel), GELU activation, and dropout (0.1) for feature extraction.
Classifier Heads	Attribute-specific, consisting of: - **Spatial Attention Branch**: Two convolutional layers (3 × 3 kernel, groups = 32), GELU activations. - **Channel Attention Branch**: Adaptive average pooling, linear layers (512 ⟶ 128 ⟶ 512), and sigmoid activation. - **Final Classification Branch**: Adaptive average pooling, flattening, linear layers (512 ⟶ 1024 ⟶ 512 ⟶ num_classes), layer normalization, GELU activation, dropout (0.1-0.2).

Final Model and Hyperparameters:

- **Model Details**: The architecture consisted of a ConvNeXT base model with specialized classifier branches for different attributes. The model size varied depending on the category. For the **'Men's T-Shirt'** category, it was the smallest, with 95,658,253 parameters and a model size of **382.81 MB**, requiring **364.91 MB** of GPU memory. On the other hand, the

'Sarees' category had the largest model, with 102,711,728 parameters, a model size of **411.17 MB**, and GPU usage of **391.82 MB**

- **Final Hyperparameters**:
  - **Learning Rate**: Dynamic scheduling using "ReduceLROnPlateau".
  - **Batch Size**: 4 images per batch.
  - **Optimizer**: AdamW.
  - **Loss Function**: Combined Cross-Entropy and MMD (Maximum Mean Discrepancy) loss during initial model training, with MMD weighted by a lambda factor of 0.1. Attribute-specific models were trained with Cross-Entropy loss only.

Hyperparameter Tuning:

- Learning rates were adjusted with a scheduler to reduce when performance plateaued, ensuring optimal convergence.

4. Novelty and Innovation

Key strengths of our solution include:

1. **Memory Efficiency:** The model has a relatively small weight size, ranging from 391 MB to 411 MB based on the category, making it lightweight compared to other models. The inference time is also significantly faster, processing 9.4 to 10.4 images per second, which is much higher than the typical required processing time for similar tasks.
2. **Novel Training Approach:** Our training approach, which combines MMD loss (for domain adaptation) with training the model in two phases, offers a unique advantage. First, we train the entire model, including the pre-trained base model, shared features, and classifier heads. This ensures that the model learns important general features. Then, we fine-tune only the classifier heads for each attribute, preventing data loss due to missing features and improving the model's generalization capability.
3. **Custom Architecture:** The architecture itself was custom-designed, incorporating spatial and channel attention mechanisms, alongside a shared backbone with category-specific classifiers. This allowed us to specialize the model for each category while maintaining efficiency.

5. Training Details

Environment: The model training was conducted on Kaggle’s infrastructure, using dual NVIDIA T4X2 GPUs.

Training Time: Each complete training session varied between 8 to 12 hours, depending on the category. The entire training process was conducted three times for each category:

- First training: Full model training, achieving a 73.807 F1-score on test data.
- Second training: Fine-tuning only the attribute classifier heads, reaching a 76.263 F1-score.
- Third training: Further fine-tuning of the classifier heads, achieving a final F1-score of 76.660.

Note: These results are corresponding to the public leaderboard.

The total training time, with a maximum of 12 hours per session, amounted to:

- Maximum training time per category: 12 hours/session × 3 sessions = 36 hours.
- Total training for all five categories: 36 hours × 5 categories = 180 hours.

Inference Speed: The model demonstrated an efficient inference capability, processing **9.4 to 10.4 images per second**, depending on the number of attributes (9.4 images/second for 10 attributes and 10.4 images/second for 5 attributes). This speed is **five times the minimum required inference speed**, which is 2 images per second (500 ms per image). The reported inference time encompasses the complete pipeline, including image resizing, pixel normalization, label encoding/decoding, and processing one image at a time. Batch-processing images might further reduce the time requirements.

6. Evaluation Metrics

Category	Micro-F1-score	Macro-F1-score	Harmonic mean of micro and macro f1 score
Men Tshirts	0.972	0.967	0.969
Sarees	0.754	0.534	0.6252
Kurtis	0.929	0.888	0.908
Women Tshirts	0.901	0.766	0.828
Women Tops & Tunics	0.913	0.854	0.882

7. Other Experiments we tried

- We filled the NaN values in a data frame by leveraging similarity among attributes. Rows with more than a specified number of NaNs (defined by the max_nans parameter, set to 3) are skipped. For each row, missing values are filled by filtering rows in the DataFrame that match the non-missing values of subsequent attributes in similar rows. These matching rows form a subset, and the most frequent value (mode) from this subset is used to fill the missing value. If no matching subset is found, the overall mode of the column serves as a fallback.
- We also experimented with replacing our base model, ConvNeXt, with Google/Bit-50 and ConvNeXt2. However, we found that ConvNeXt, when used with the removed NaN values, performed better, so we decided to continue with it.

8. Conclusion

Our model achieved a ranking of **23rd** on the private leaderboard with an F1-score of **76.93%**, just 3.27% behind the top-performing team. Despite this, our solution stands out due to its deployment-friendly nature, requiring minimal computational resources and exhibiting fast inference times. The approach ensures efficient memory usage and rapid processing, making it an ideal candidate for real-world applications.