Track 4: Text-Guided Image Clustering

Ananya Sharma
Harshit Joshi

Department of Mathematics
Indian Institute of Technology, Delhi

April 18, 2025

Outline

Data Pre-Processing and Preparation

- **Sampling:** Randomly sampled 10% of images per class.
  1. **Dataset:** Food-101
     - $N = 2100$ images total
     - $C = 30$ classes
     - 70 images per class
  2. **Dataset:** ImageNet
     - $N = 1500$ images total
     - $C = 30$ classes
     - 50 images per class
- **Reproducibility:** Fixed random seed=782 used (for sampling and K-Means initialization)

Non-Competitive Section

Classical Feature Extraction

Traditional computer vision features are used to cluster the images.

Data Pre-Processing and Preparation

- All images I from the $N = 2100$ sample were first resized to 224×224 pixels using OpenCV's `cv2.resize`, resulting in an 8-bit integer image $I' \in \mathbb{Z}^{224 \times 224 \times 3}$ with pixel values between 0 and 255.

I: Original

I′: Resized 8-bit

- **SIFT** and **Canny edge detection** operated on the 8-bit image I' and its grayscale conversion respectively.

Data Pre-Processing and Preparation

- For features like **Color Histograms**, **LBP**, and **HOG**, we normalized the pixel values to the range $[0, 1]$ by converting to float and dividing by 255. This normalized image is $\tilde{I} \in [0, 1]^{224 \times 224 \times 3}$.

$\tilde{I}$: Normalized [0,1]

I_{gray}: Grayscale

SIFT

- Detects keypoints and extracts 128-dimensional descriptors from local patches around those keypoints in the image.
- Divided the image into 4 spatial quadrants: **TL (Top Left), TR (Top Right), BL (Bottom Left), BR (Bottom Right)**.
- For each quadrant, collect descriptors whose keypoints fall inside that region. Compute the mean of all descriptors in that quadrant.
- If a quadrant has no keypoints, fill in a zero vector of 128D. Concatenate the four 128D vectors.

SIFT

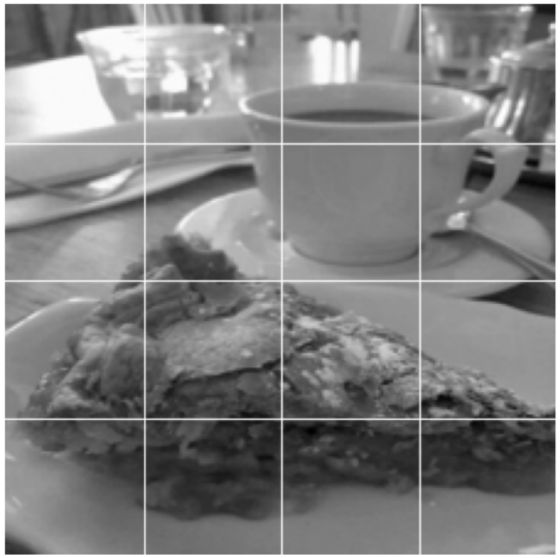
- The final SIFT representation of the image is a single 512-dimensional vector, $f_{\text{SIFT}} = [\mu_{TL}^T, \mu_{TR}^T, \mu_{BL}^T, \mu_{BR}^T]^T \in \mathbb{R}^{512}$

$$\Rightarrow \quad \begin{bmatrix} x_1^{TL} \\ x_2^{TL} \\ \vdots \\ x_{128}^{TL} \\ \vdots \\ x_{128}^{BR} \end{bmatrix} \in \mathbb{R}^{512}$$

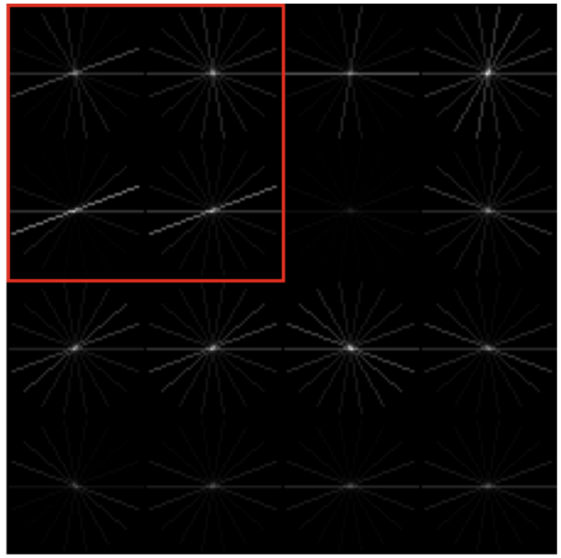
Histogram of Oriented Gradients (HOG)

- As only intensity is essential for capturing edge directions so we convert the image to grayscale.
- The image is divided into cells of size 56×56 pixels. So, we get 4 cells along each dimension. Total cells = 4×4 = 16.
- Each block is a 2×2 group of adjacent cells and blocks overlap. So, no. of blocks = (4-2+1)×(4-2+1) = 9 blocks
- For each cell, compute a histogram of gradient directions with 9 bins. Each cell gives a 9-dimensional vector.
- So, for a 2×2 block, we get 4×9 = 36 features.
- There are 9 blocks. So, total features = 324 $i.e. f_{\text{HOG}} \in \mathbb{R}^{324}$

HOG



$\Rightarrow$



$\Rightarrow$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{324} \end{bmatrix} \in \mathbb{R}^{324}$$

Color Histogram

Method:

- Normalize image by scaling pixel values for each channel (Red, Green, Blue) to the range [0, 1].
- Compute 32-bin histograms for each channel and concatenate the three vectors into a single 96-dimensional vector
  $i.e. f_{\text{Color}} = [h_B^T, h_G^T, h_R^T]^T \in \mathbb{R}^{96}$

Canny Edge Detection

Method:

- Edge detection is intensity-based, so we convert the color image to grayscale.
- Apply Canny edge detection with thresholds 200 and 100.
- Divide Edge Map into an 8×8 Grid
- Compute normalized edge density per cell.
  Edge density $= \frac{\text{Sum of edge pixels}}{255 \times \text{number of pixels in the cell}}$
- Stack all 64 cell densities into a vector we get $f_{\text{Edge}} \in \mathbb{R}^{64}$.

Canny Edge Detection

$\Rightarrow$

$\Rightarrow$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{64} \end{bmatrix} \in \mathbb{R}^{64}$$

Local Binary Patterns (LBP)

Method:

- We compute uniform LBPs ($P = 8, R = 1$) on grayscale $\tilde{I}$ image using `skimage.feature.local_binary_pattern`.
- The normalized histogram of the 10 uniform patterns gave $f_{LBP} \in \mathbb{R}^{10}$.

Concatenated Classical Features

Concatenated all classical features:

$$\mathbf{f}_{\text{concat}} = [\mathbf{f}_{\text{SIFT}}^T, \mathbf{f}_{\text{HOG}}^T, \mathbf{f}_{\text{Color}}^T, \mathbf{f}_{\text{Edge}}^T, \mathbf{f}_{\text{LBP}}^T]^T \in \mathbb{R}^{1006}.$$

Time Taken

- Feature extraction process took about 3 minutes for the 2100 images.

Results

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the features extracted from all the 2100 images. The Adjusted Rand Index (ARI) scores are shown below:

Technique	K-Means ARI	DBSCAN ARI
SIFT	0.0210	0.0
HOG	0.0145	0.0
Color Histogram	0.0076	0.0
Canny Edge Histogram	0.0198	0.0060
LBP	0.0060	0.0
Concatenated	0.0077	0.0

Results (ImageNet)

Technique	K-Means ARI	DBSCAN ARI
SIFT	0.0171	0.0
HOG	0.0325	0.0
Color Histogram	0.0213	0.0
Canny Edge Histogram	0.0320	0.0005
LBP	0.0171	0.0
Concatenated	0.0208	0.0

Deep Feature Extraction

Features extracted using pretrained ResNet-50.

ResNet-50

Preprocessing:

- We use the standard ImageNet preprocessing pipeline via `torchvision.transforms`:
  - Resize to 256×256
  - Center crop to 224×224
  - Convert to PyTorch tensor I''
  - Normalize using ImageNet mean $\boldsymbol{\mu} = [0.485, 0.456, 0.406]$ and std $\boldsymbol{\sigma} = [0.229, 0.224, 0.225]$
- The normalized image I_{norm} is computed as:

$$I_{\text{norm}} = \frac{I'' - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

ResNet-50

Original I

$\Rightarrow$

I_{norm}

ResNet-50

Features Extracted:

- We took the output from the penultimate layer (final average pooling layer, before the original classification layer) by replacing the final layer with an identity map resulting in $f_{\text{ResNet50}} \in \mathbb{R}^{2048}$.

Time Taken

- Feature extraction for ResNet-50 took about 15 minutes for 2100 images.

ResNet-50

Results

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the extracted ResNet-50 features.

Technique	K-Means ARI	DBSCAN ARI
ResNet-50 (Pretrained)	0.2285	0.0

For ImageNet

Technique	K-Means ARI	DBSCAN ARI
ResNet-50 (Pretrained)	0.5028	0.0

Text-Guided Feature Extraction

Features extracted by applying SBERT on the text captions generated using BLIP.

BLIP Captions

Preprocessing

- Image preprocessing was handled internally by `transformers` BLIP model pipeline.

Features Extracted

- **Caption Generation:** We used the BLIP base model (`Salesforce/blip-image-captioning-base`) to generate a caption c_i for each image I_i.
- **Text Embedding:** We encoded the captions $\{c_i\}$ using Sentence-BERT (`all-mpnet-base-v2`) into vectors $f_{\text{BLIP+SBERT}} \in \mathbb{R}^{768}$.

BLIP Captioning

$\Rightarrow$ *'a plate with a piece of pie on it'*

Generating Text Descriptions and Features

NOTE: The quality of generated captions varied.

- **Some descriptive:** "a plate with a hamburger and fries on it"
- **Some vague:** "a plate of food on a table"
- **Some contained repetitions:** "two taco ta ta ta ta..."

Time Taken

- Caption generation took approx. 10 mins for 2100 images. SBERT embedding time was negligible in comparison.

Generating Text Descriptions and Features

Results

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the SBERT embeddings of the BLIP-generated captions.

Technique	K-Means ARI	DBSCAN ARI
BLIP Captions + SBERT	0.2115	0.0140

For ImageNet

Technique	K-Means ARI	DBSCAN ARI
BLIP Captions + SBERT	0.2188	0.0006

Competitive Section

Alternative CNN architectures

Features extracted using pretrained EfficientNet-B0.

EfficientNet-B0

As an alternative to ResNet-50, we extracted baseline features using EfficientNet-B0 which is another CNN pretrained on ImageNet, but much more lightweight.

Preprocessing

- The preprocessing was identical to that used for ResNet-50.

Features Extracted

- The extracted features were the output of the `model.avgpool` layer, $f_{\text{EffNetB0}} \in \mathbb{R}^{1280}$.

Time Taken

- Feature extraction took about 6 minutes for 2100 images.

EfficientNet-B0

Results

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the pretrained EfficientNet-B0 features.

Technique	K-Means ARI	DBSCAN ARI
EfficientNet-B0 (Pretrained)	0.2791	0.0

Alternative Text architectures

Using VQA Models to generate text.

VILT VQA + SBERT

Now we explored using Visual Question Answering (VQA) as an alternative text generation method for feature extraction.

Preprocessing:

- Image preprocessing is handled automatically by the **VILTProcessor** from HuggingFace's transformers library.

Features Extracted:

- **VQA Generation:** We used the VILT model (`dandelin/vilt-b32-finetuned-vqa`) to answer the question `"Which dish is in this image ?"` for each image I_i, yielding an answer a_i.
- **Text Embedding:** We encoded answers $\{a_i\}$ using SBERT ('all-mpnet-base-v2') into $f_{\text{VILT+SBERT}} \in \mathbb{R}^{768}$.

VILT VQA + SBERT

NOTE: The answers were very cryptic and contained special tokens (e.g., “[unused295]”, “”, “”, “”), possibly due to the model’s original training or limitations in generating specific dish names in response to our question.
NOTE: We wanted the question to be more specific; however, the sequence length limit was of 40 words, which made it difficult to specify all of the classes while being fair to all.

VILT VQA + SBERT

Time Taken

- VQA generation took approx. 25 mins for 2100 images. SBERT embedding time was negligible.

Results

- We clustered the SBERT embeddings of the VILT VQA answers using K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`).

Technique	K-Means ARI	DBSCAN ARI
VILT VQA + SBERT	0.1699	0.0426

Fine-tuning Deep Features

Making EfficientNet-B0 specialized on our data domain

Fine-tuning Deep Features (EfficientNet-B0)

We wanted our model to be familiar with our data domain, so we fine-tuned EfficientNet-B0 on our specific food dataset subset. The motivation was to make the model more specialized.

Preprocessing

- We split the 2100 images into balanced train/test sets (1050 each, 35 per class) using the same random seed (`seed = 782`).
- Preprocessing for training and testing used the same ImageNet pipeline as the pretrained model (resize 256, crop 224, ToTensor, normalize), for training we also include `RandomHorizontalFlip`
- We replaced the final classifier of a pretrained EfficientNet-B0 with a new linear layer matching our 30 food classes.

Fine-tuned EffNet-B0

Feature Extraction:

- The base model is EfficientNet-B0. The final layer is replaced for fine-tuning on 30 classes.
- After fine-tuning, features $f_{\text{EffNetB0-FT}} \in \mathbb{R}^{1280}$ were extracted from the output of the `model.avgpool` layer of the fine-tuned model.

Training and time taken

- **Training:** We trained the adapted model for 15 epochs using 'CrossEntropyLoss' and the Adam optimizer (lr=1e-4). The model achieved a test accuracy of approximately 71%. Training took approx. 6 mins for the 1050 **train** images.
- **Feature Extraction:** We used the fine-tuned model (loading the saved state) to extract features from the 1050 **test** images. Feature extraction took approx. 1 min.

Fine-tuned EffNet-B0

Results and inferences

- We clustered the features extracted from the 1050 test images using K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`).

Technique	K-Means ARI	DBSCAN ARI
EfficientNet-B0 (Fine-tuned)	0.4540	0.0

Fine-tuning Text Features

Fine-Tuning BLIP for captions to be more specific

Fine-tuning Text-Guided Features (BLIP Captions)

We explored if fine-tuning the BLIP image captioning model on our dataset could enhance the quality of text-based features for clustering.

Preprocessing

- We used the same 50/50 train/test split (1050 images each) as for fine-tuning EfficientNet-B0 (`seed = 782`).
- For training, we paired images with simple template captions: "This is an image of [class_name]" (replaced all '_' with ' ').
- Image preprocessing for BLIP fine-tuning is handled by its specific processor.

BLIP Captioning

$\Rightarrow$ *'This is an image of apple pie'*

Fine-tuned BLIP

Feature Extracted

- **Caption Generation (Fine-tuned):** We generated captions c_i' for the 1050 **test** images using the fine-tuned BLIP model (‘Salesforce/blip-image-captioning-base‘ adapted).
- **Text Embedding:** We encoded these new captions $\{c_i'\}$ using the same SBERT model (‘all-mpnet-base-v2‘) into vectors $\mathbf{f}_{\text{BLIP-FT+SBERT}} \in \mathbb{R}^{768}$.

Fine-tuned BLIP

Training and time taken

- **Training:** We fine-tuned the ‘Salesforce/blip-image-captioning-base‘ model for 5 epochs using the Hugging Face ‘Trainer‘ library (configuration: batch size 4, gradient accumulation steps 2, learning rate 5×10^{-5}, mixed-precision fp16). Training took approx. 20 mins for the 1050 **train** images.
- **Time Taken:** Caption generation using the fine-tuned model and subsequent SBERT embedding were performed on the 1050 test images. Time taken was about 5 minutes for the caption generation.

Fine-tuned BLIP

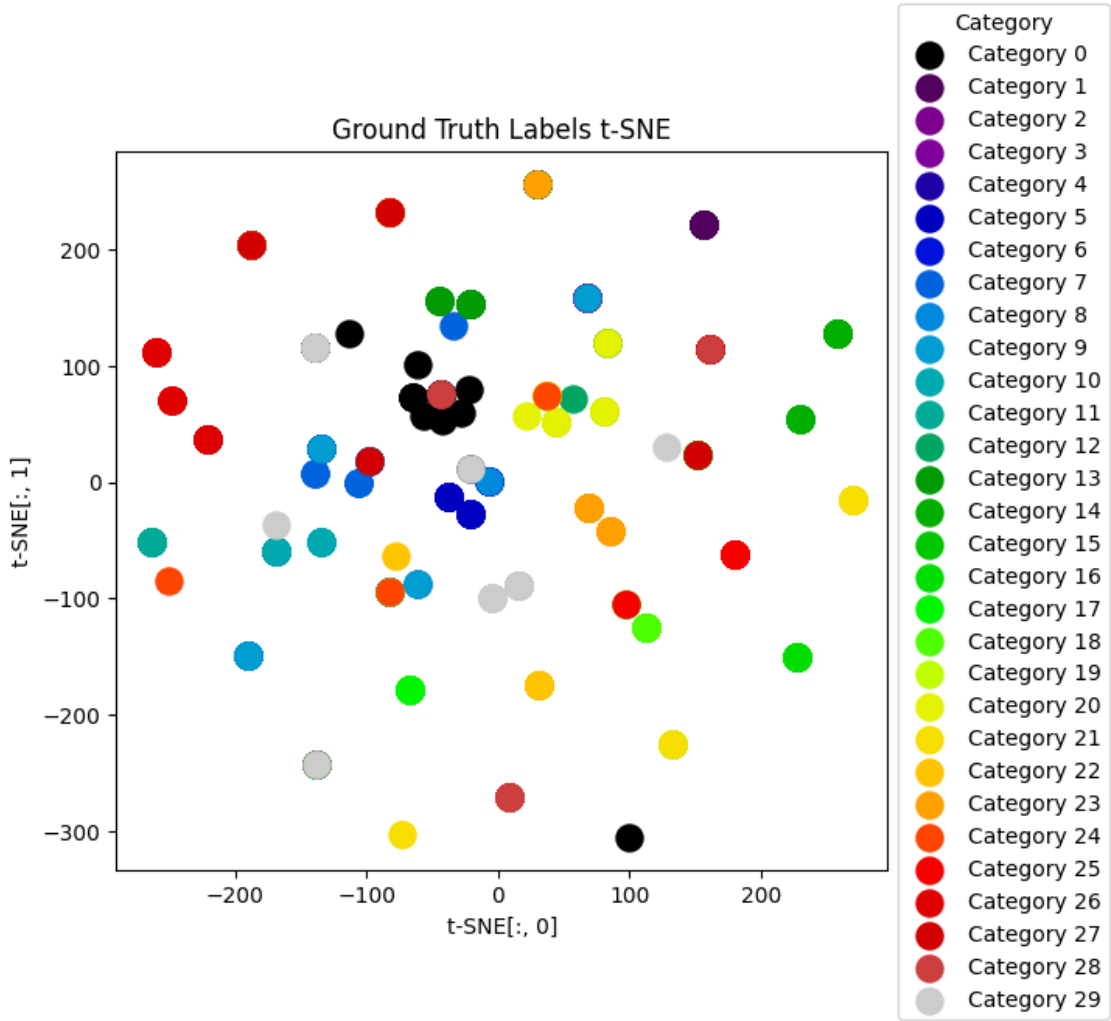
Results

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the SBERT embeddings derived from the fine-tuned BLIP captions (using the 1050 test images).

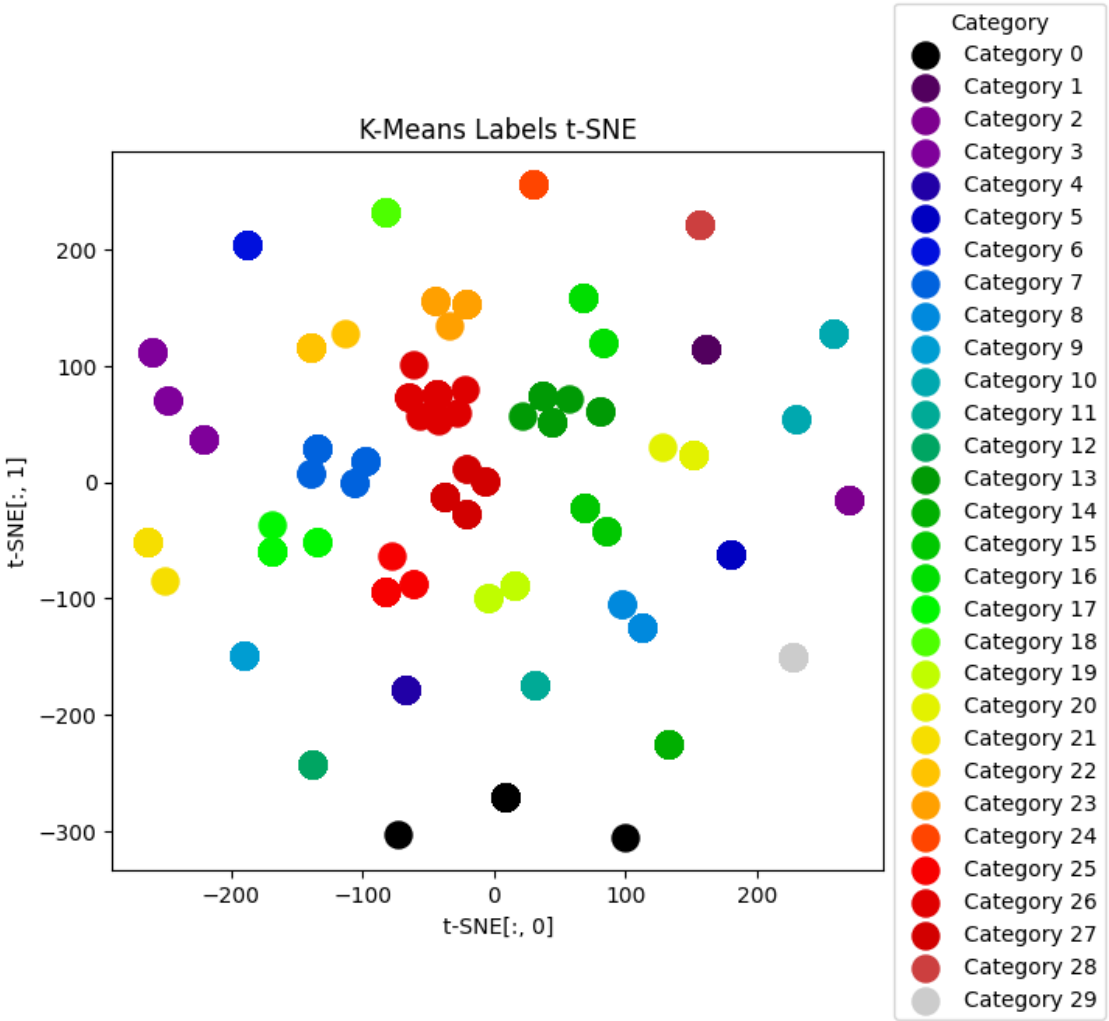
Technique	K-Means ARI	DBSCAN ARI
BLIP (Fine-tuned) + SBERT	0.7726	0.7741

Fine-tuned BLIP: t-SNE Visualization (Ground Truth)

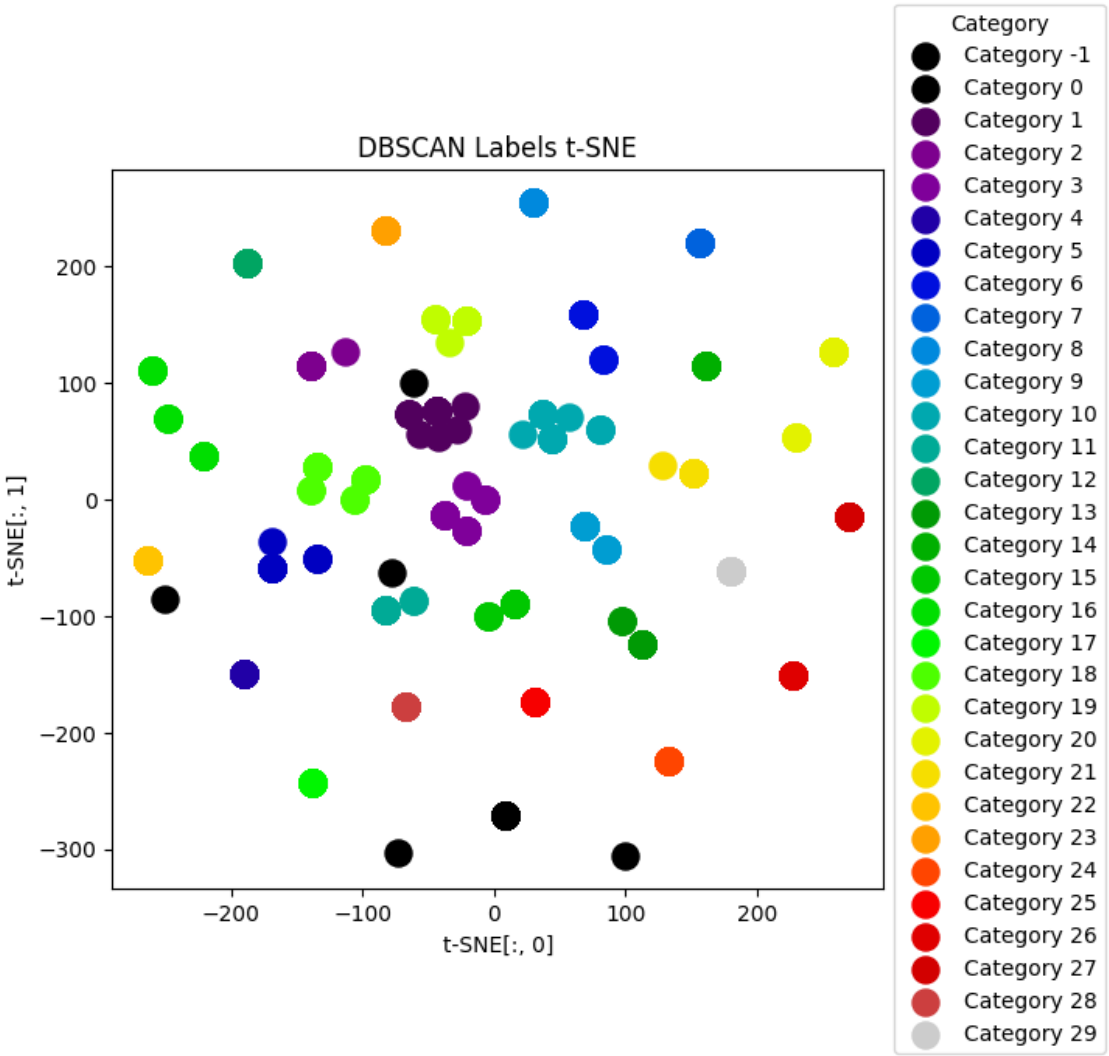
As this is the first approach which produced promising results, we visualized the features using `t-SNE`.



Fine-tuned BLIP: t-SNE with K-Means Labels



Fine-tuned BLIP: t-SNE with DBSCAN Labels



Note: DBSCAN flags a few points as noise (label −1).

Dimensionality Reduction

Using simple NN to compress the embedded features

Dimensionality Reduction via Neural Network

We wanted to investigate whether compressing the highly effective fine-tuned BLIP+SBERT features ($f_{\text{BLIP-FT+SBERT}} \in \mathbb{R}^{768}$) using a neural network could enhance clustering performance, potentially by isolating the most discriminative information.

Preprocessing

- The input data for this step consisted of the 1050 fine-tuned BLIP+SBERT feature vectors ($f_{\text{BLIP-FT+SBERT}} \in \mathbb{R}^{768}$) generated from the **test** set images in Section 4.4.
- To split these embeddings using an 80/20 split, so:
  - Training set: 840 embeddings (28 per class)
  - Test set: 210 embeddings (7 per class)

Dimensionality Reduction

Feature Extraction

- We designed a simple feed-forward neural network (CompressNet) with an architecture of $768 \rightarrow 128 \rightarrow 30$. The intermediate 128-dimensional layer is taken as the compressed embedding space.
- After training the network on the 840 **train** images, we processed the fine-tuned BLIP+SBERT features from the 210 **test** images. Let these features be $f_{\text{Compressed}} \in \mathbb{R}^{128}$.

Training and time taken

- **Training:** The CompressNet was trained for 20 epochs using 'CrossEntropyLoss' and an Adam optimizer.
- **Time Taken:** Training took about 5 seconds. Feature extraction time for the test set was negligible.

Dimensionality Reduction

Results and inferences

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the extracted 128-dimensional features $f_{Compressed}$ from the test set.

Technique	K-Means ARI	DBSCAN ARI
Compressed BLIP-FT+SBERT	0.7760	0.7741

Final Model

Final Model

For the final clustering of the images provided in Kaggle Test Set, we decided to train BLIP using a larger portion of the available data (to potentially further improve the fine-tuned model's performance) and then use it to generate captions and then encode them using SBERT for the final evaluation.

Preprocessing

- Instead of the initial 10% sample (2100 images), we sampled 20% of the images from each of the 30 classes in the given dataset. This resulted in a dataset of 4200 images (140 images per class).
- This new dataset (4200 images) was split 80% for training (3360 images, 112 per class) and 20% for testing (840 images, 28 per class).

Final Model: Fine-tuned BLIP + SBERT

Training and time taken

- **Training:** The 'Salesforce/blip-image-captioning-base' model was fine-tuned on this larger training set (3360 images) for 5 epochs using the same template captions ("This is an image of [class_name]") and hyperparameters as described previously.
- **Time Taken:** The training took about 40 minutes to complete

Results

- We applied K-Means ($k = 30$, `random_state=782`) and DBSCAN (`eps=0.5`, `min_samples=10`) to the extracted 768-dimensional features from the **test** set.

Technique	K-Means ARI	DBSCAN ARI
Final BLIP-FT+SBERT	0.8164	0.8164

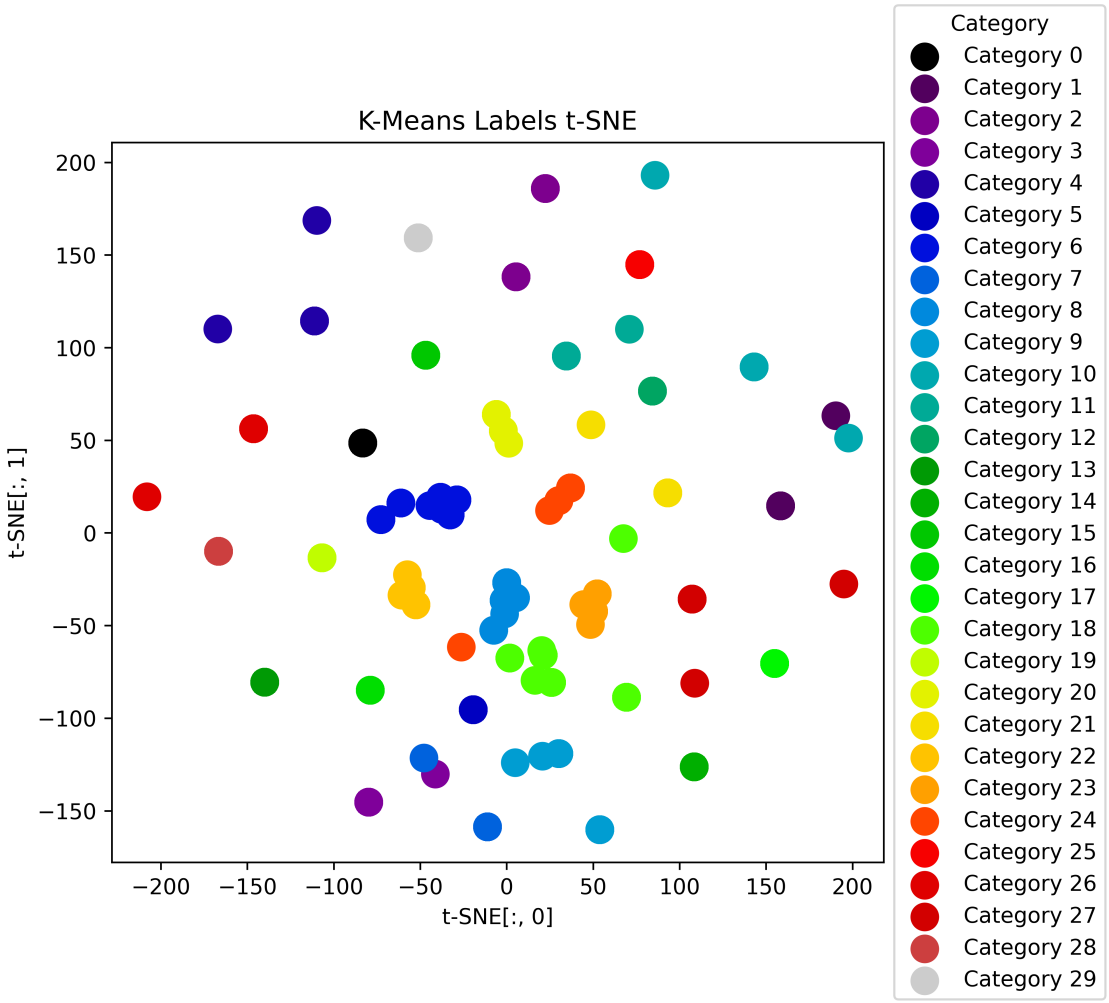
Applying to Kaggle Test Dataset Food-101

- We now used this model on the provided test dataset, it consisted of 9000 images, generating captions for which took about 30 minutes
- After using SBERT to encode them to 768D features we applied three different clustering algorithms.
- The ARI scores achieved were:

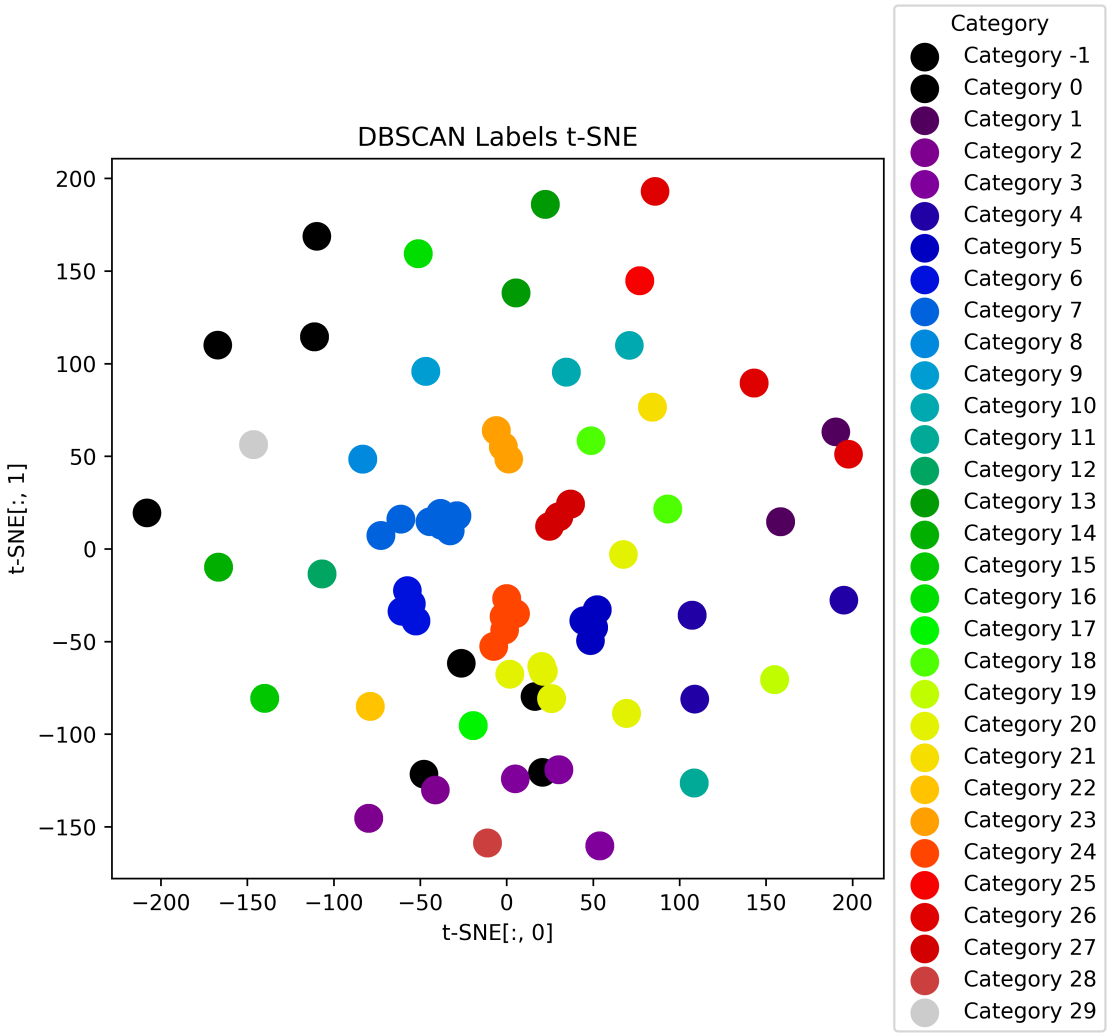
Algorithm	Kaggle Private ARI
K-Means (k=30, seed=782)	0.8036
DBSCAN (eps=0.5, min_samp=10)	**0.8041**
Agglomerative Clustering (k=30)	0.8036

t-SNE Visualization

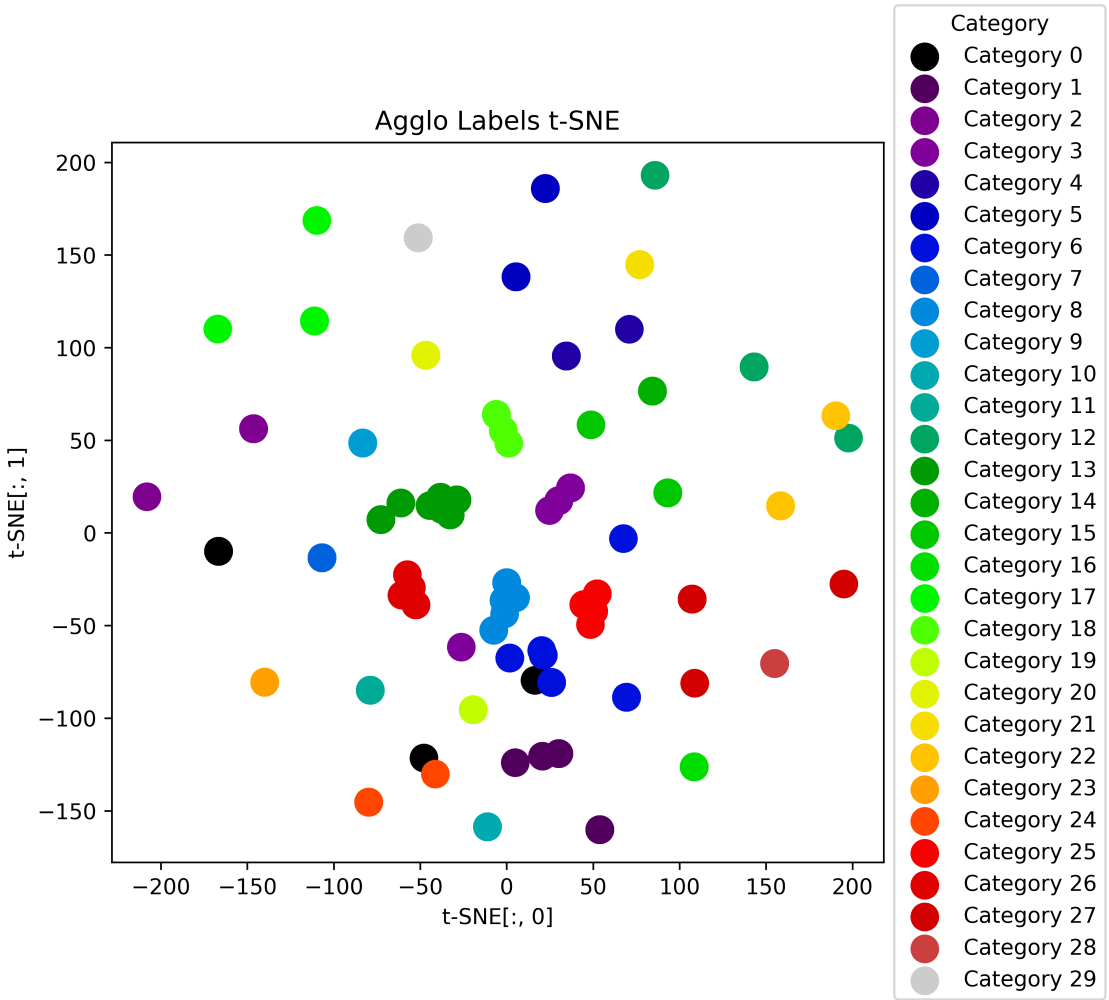
Visualizing the $\mathbb{R}^{768}$ features for the 9000 Kaggle test images using t-SNE with predicted labels.



t-SNE Visualization



t-SNE Visualization



Note: DBSCAN flags a few points as noise (label −1).

Thank You