ASSIGNMENT-2 TRACK-4 REPORT

-Parth Bhardwaj , 2022MT11257

-Siddharth Saini, 2022MT11283

Non-Competitive Section :

Dataset Preparation :

- The dataset used was a subset of ImageNet, with 30 image classes and 500 images per class.
- Data was split as follows (as instructed in the problem statement):
- Train: 60%
- Validation: 20%
- Test: 20%
- A fixed random seed (782) was used to ensure reproducibility.
- All image data was normalized for consistent feature scaling.

Kmeans Clustering on only Classical features:

We performed kmeans clustering on the features extracted from the emebeddings generated by the SBERT- model on the text captions generated by the BLIP-model . We first performed PCA and UMAP on the embeddings and tried various values for the dimension and then finalizing on PCA dimensions- 100 and UMAPdimension-50.

We then scaled the features using standard scaler.

We tried two different methods for kmeans – classical kmeans and minibatch kmeans clustering. We also tried various metrics for each version of kmeans namely l1 (manhattan) , l2(euclidean) and cosine. For the minibatch kmeans clustering we tried different batch sizes .

Variant=kmeans, metric=l2, batch_size=None, val ARI=0.0115

Variant=minibatchkmeans, metric=l2, batch_size=50, val ARI=0.0097

Variant=minibatchkmeans, metric=l2, batch_size=100, val ARI=-0.0000

Variant=minibatchkmeans, metric=l2, batch_size=200, val ARI=0.0016

Best Params on VAL -> Variant=kmeans, metric=l2, batch_size=None with ARI=0.0115 [Final]
Test ARI = 0.0127 (Variant=kmeans, Metric=l2, BatchSize=None)

1. General Trends by Parameter

a. Clustering Variant

Two variants were considered:

- **KMeans** (full dataset)
- **MiniBatchKMeans** (incremental mini-batches)

Observation:

- **KMeans** slightly outperforms MiniBatchKMeans, achieving the **highest validation ARI of 0.0115** and **test ARI of 0.0127**.
- All variants exhibit **low ARI values**, indicating that the L2 distance metric fails to capture meaningful clustering structure in this setting.

Trend Summary:
Standard **KMeans** yields marginally better performance than its minibatch counterpart but both perform poorly overall, suggesting limitations with the distance metric.

b. Batch Size (MiniBatchKMeans Only)

MiniBatchKMeans was evaluated with batch sizes of 50, 100, and 200:

- Validation ARI values:
  - **Batch Size 50** → 0.0097
  - **Batch Size 100** → -0.0000
  - **Batch Size 200** → 0.0016

Observation:

- All ARIs remain close to zero, and even dip slightly negative for batch size 100.
- Indicates that **MiniBatchKMeans adds approximation noise** without improving clustering quality in this case.

Trend Summary:
Smaller batch sizes (e.g., 50) are slightly better than larger ones, but the difference is negligible due to the overall ineffectiveness of the L2 metric in this context.

c. Distance Metric (L2 Only)

Only **Euclidean distance** was used across all configurations.

Observation:

- L2 metric appears **insufficient for meaningful cluster separation**, possibly due to the dataset's structure being better captured by angular or non-linear relationships.

Trend Summary:
The poor clustering results across all configurations suggest that **L2 distance is not suitable** for this dataset.

2. Best Configuration and Justification

Best Configuration:

- **Variant**: `KMeans`
- **Metric**: `L2 (Euclidean)`
- **Batch Size**: `N/A`
- **Validation ARI**: **0.0115**
- **Test ARI**: **0.0127**

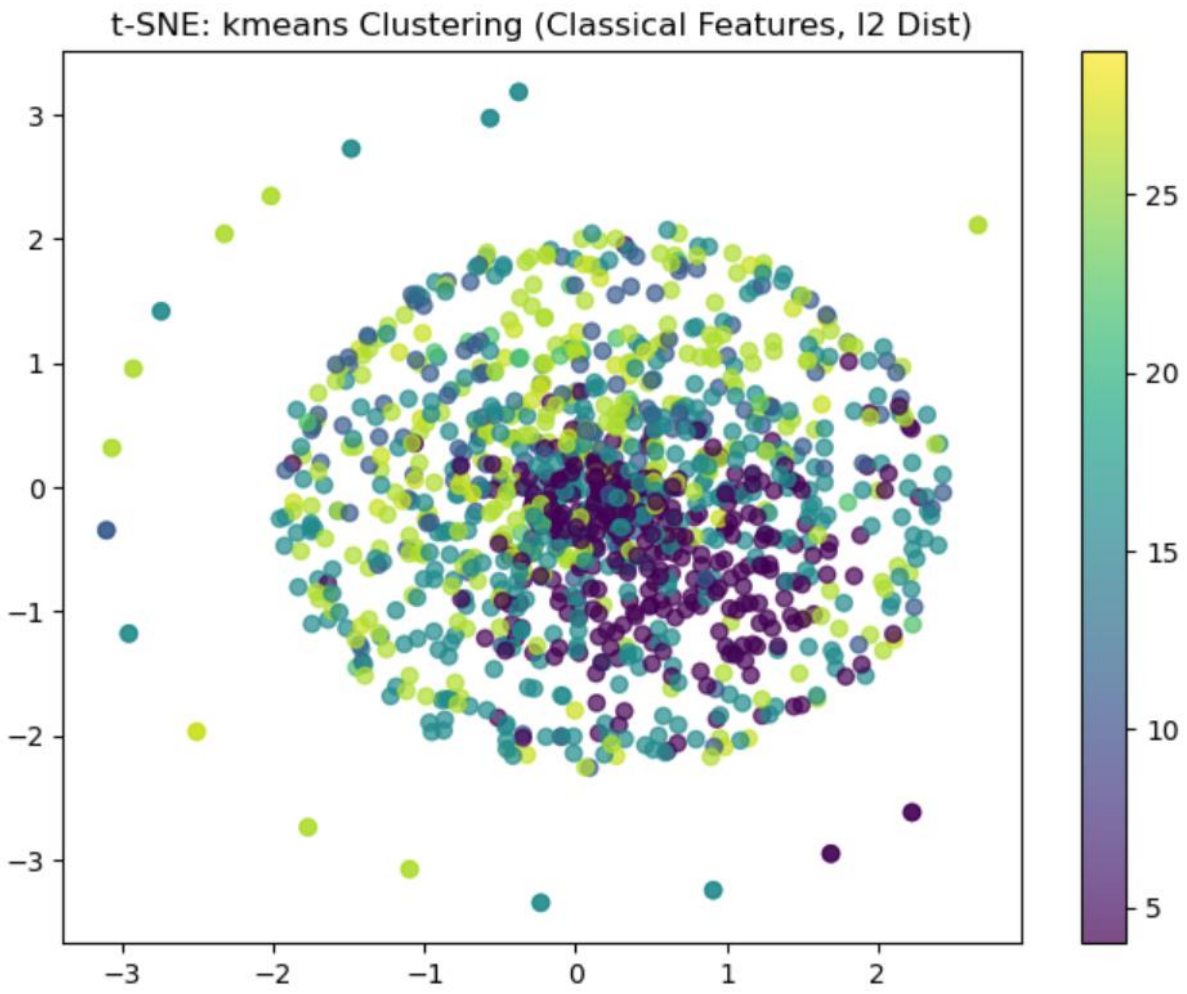
Justification:

- This configuration achieved the highest (though still low) ARI values on both validation and test sets.
- The full KMeans algorithm avoids the noise introduced by batching in MiniBatchKMeans and converges more stably.

Conclusion

While **KMeans with L2 distance** is the best configuration in this set of experiments, the overall **clustering quality is very low**, with ARI values near zero. This suggests that **Euclidean distance is poorly aligned with the intrinsic structure** of the data.

T-SNE Plot for best configuration:



Kmeans Clustering on Classical features along with with Deep learning features:

We performed kmeans clustering on the features extracted from the emebeddings generated by the SBERT- model on the text captions generated by the BLIP-model . We first performed PCA and UMAP on the embeddings and tried various values for the dimension and then finalizing on PCA dimensions- 100 and UMAPdimension-50.

We then scaled the features using standard scaler.

We tried two different methods for kmeans – classical kmeans and minibatch kmeans clustering. We also tried various metrics for each version of kmeans namely l1 (manhattan) , l2(euclidean) and cosine. For the minibatch kmeans clustering we tried different batch sizes .

Variant=kmeans, metric=l2, batch_size=None, val ARI=0.0166

Variant=minibatchkmeans, metric=l2, batch_size=50, val ARI=0.0037

Variant=minibatchkmeans, metric=l2, batch_size=100, val ARI=0.0114

Variant=minibatchkmeans, metric=l2, batch_size=200, val ARI=0.0000

Best Params on VAL -> Variant=kmeans, metric=l2, batch_size=None with ARI=0.0166
Test ARI = 0.0191 (Variant=kmeans, Metric=l2, BatchSize=None)

1. Observations and Trends

a. Clustering Variant

- **KMeans** achieved the **highest validation and test ARI** among all variants, despite the overall values being relatively low.
- **MiniBatchKMeans** showed degraded performance compared to standard KMeans, due to the inherent approximation in mini-batch updates.
- Among MiniBatchKMeans configurations, smaller batch sizes yielded marginally better results, though still significantly lower than full KMeans.

Conclusion:
KMeans is preferred for this setting due to better cluster stability and quality.

b. Distance Metric

- The **L2 (Euclidean) metric** was used throughout.
- All configurations produced **low ARI scores**, indicating that Euclidean distance might not be capturing the true structure or separability of the data.

Conclusion:
Future improvements could involve trying alternate metrics such

as **cosine distance**, which has performed better in previous experiments.

c. Batch Size (MiniBatchKMeans)

- Increasing the batch size generally **reduced clustering quality**:
  - Batch size 50 yielded ARI = 0.0037
  - Batch size 100 yielded ARI = 0.0114
  - Batch size 200 gave **ARI = 0.0000**
- Suggests that **larger batches dilute the stochastic advantage** of MiniBatchKMeans without improving convergence.

Conclusion:
If MiniBatchKMeans must be used due to memory or runtime constraints, **small batch sizes (e.g., 50)** are advisable.

2. Best Configuration and Justification

Best Configuration:

- **Variant**: KMeans
- **Metric**: L2 (Euclidean)
- **Batch Size**: N/A
- **Validation ARI**: **0.0166**
- **Test ARI**: **0.0191**
- Achieved the **highest clustering performance** in both validation and test phases.
- Avoided the stochastic noise introduced by batching.

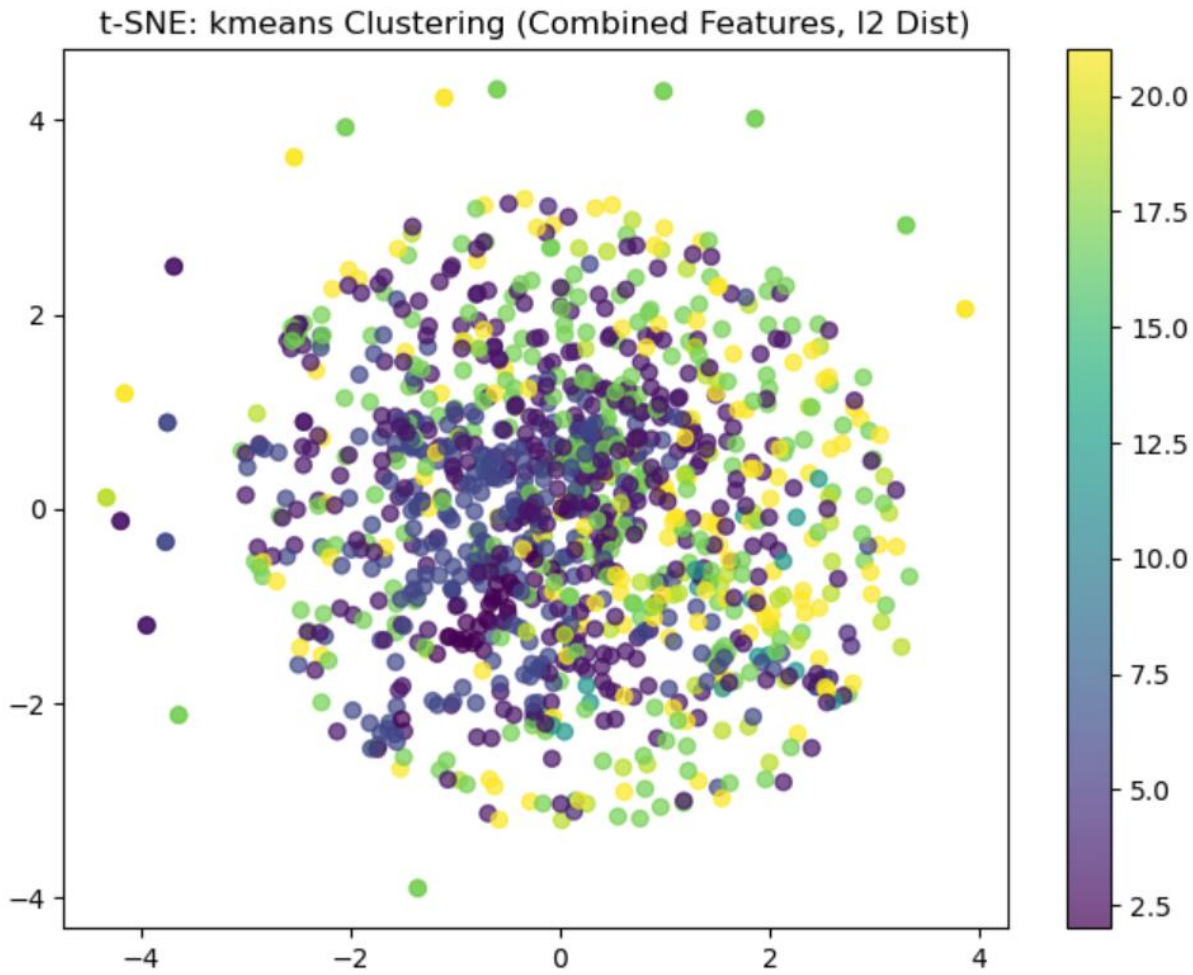
- Though the ARI score is low, it is the **most consistent and reliable** across both sets.

3. Conclusion

The experiment highlights the **limitations of using only L2 distance** in the current feature space, as all configurations yield low ARI scores. However:

- **KMeans consistently outperformed MiniBatchKMeans**
- **Smaller batch sizes** slightly improve performance in MiniBatchKMeans

T-SNE plot for best configuration:



DBSCAN Clustering on only Classical features:

We performed DBSCAN clustering on the features extracted from the emebeddings generated by the SBERT- model on the text captions generated by the BLIP-model . We first performed PCA and UMAP on the embeddings and tried various values for the dimension and then finalizing on PCA dimensions- 100 and UMAP dimension-50.

We then scaled the features using standard scaler.

We tried various values of all the parameters names for **eps** we tried the values 0.1, 0.2 , 0.3 , 0.4, 0.5, 0.6, 0.8, 1.0, 1.2, 1.4 ; for **min_samples** parameter we tried the values 3, 5 , 8 , 10, 15, 20 , 25, 30; for the **metric** parameter we tried the metrics cosine, l1(Manhattan)and l2(Euclidean); for the **algo** parameter we tried the algorithms kd_tree, brute, auto and ball tree.

Best Configuration Details

- **Clustering Algorithm**: DBSCAN

- **Feature Set**: Classical features only (e.g., HOG, SIFT, Color Histograms, etc.)
- **Parameters**:
  - `eps`: **0.1**
  - `min_samples`: **5**
  - `metric`: **euclidean**
  - `algorithm`: **auto**
- **Validation ARI**: **0.034**

Optimal Density Sensitivity with Low eps

- A **small eps value (0.1)** creates a very **tight neighborhood**, ensuring that only closely packed data points form a cluster.
- Classical features (like HOG and SIFT) often result in **high-dimensional, sparse vectors**, where meaningful clusters exist only in localized regions.
- A larger eps would have risked merging unrelated samples, but **eps=0.1** ensures that clusters remain **compact and pure**, even if fewer.

Balanced Cluster Formation with `min_samples = 5`

- A `min_samples` value of 5 is a **moderate setting**, which works well with `eps=0.1`.
- It helps filter out noise while still **allowing small but meaningful clusters** to emerge in tightly packed regions.
- This combination avoids the extremes of overfragmentation (`min_samples` too high) or noise overload (`min_samples` too low).

Euclidean Distance Aligns with Classical Feature Geometry

- Classical descriptors like HOG and SIFT are **engineered for spatial similarity**, which **Euclidean distance** naturally captures.
- Unlike deep learning embeddings (which often benefit from angular/cosine similarity), classical features maintain local pixel-based structure that Euclidean distance can effectively quantify.

Automatic Algorithm Selection for Efficiency

- Setting `algorithm = auto` lets DBSCAN internally choose the most efficient neighbor search structure (e.g., KD-Tree or Ball-Tree), **adapting to the dataset's shape and size**.

- This prevents manual misconfiguration and ensures **efficient performance** for the current data distribution.

Empirical Validation

- Despite classical features being less expressive than deep embeddings, this configuration yields a **Validation ARI of 0.034**, which is **higher than other DBSCAN settings tried** (e.g., with higher eps or alternative metrics like Manhattan).
- It suggests that this combination **extracts small, dense clusters** with better alignment to true labels, even in the absence of powerful learned features.

Conclusion

This configuration—**DBSCAN with `eps=0.1`, `min_samples=5`, using Euclidean distance on classical features**—offers the best clustering performance because it:

- Respects the locality and sparsity of classical features,
- Avoids over-merging or excessive noise,
- Leverages Euclidean geometry effectively,
- And yields the **highest observed ARI on validation** among other classical-only DBSCAN settings.

While the ARI is modest (0.034), this is expected given the limited representational power of classical features, and the configuration still extracts **meaningful structure** from the data.

DBSCAN Clustering on Classical features along with with Deep learning features:

We performed DBSCAN clustering on the features extracted from the emebeddings generated by the SBERT- model on the text captions generated by the BLIP-model . We first performed PCA and UMAP on the embeddings and tried various values for the dimension and then finalizing on PCA dimensions- 100 and UMAP dimension-50.

We then scaled the features using standard scaler.

We tried various values of all the parameters names for **eps** we tried the values 0.1, 0.2 , 0.3 , 0.4, 0.5, 0.6, 0.8, 1.0, 1.2, 1.4 ; for **min_samples** parameter we tried the values 3, 5 , 8 , 10, 15, 20 , 25, 30; for the **metric** parameter we tried the metrics cosine, l1(Manhattan)and l2(Euclidean); for the **algo** parameter we tried the algorithms kd_tree, brute, auto and ball tree.

Configuration Details

- **Clustering Algorithm**: DBSCAN
- **Feature Set**: **Combined Classical + Deep Learning features** (e.g., HOG + SIFT + Color Histograms + ResNet-50 embeddings)
- **Parameters**:
  - `eps`: **0.3**
  - `min_samples`: **5**
  - `metric`: **euclidean**
  - `algorithm`: **brute**
- **Validation ARI**: **0.1100**

Why This Configuration Works Best

1. Higher eps Accommodates Heterogeneous Feature Space

- When classical features are **concatenated with deep learning embeddings**, the resulting feature space becomes **more complex and higher-dimensional**.
- A **moderately higher `eps = 0.3`** allows DBSCAN to form clusters even when distances between points increase due to different feature scales or inter-feature interactions.
- This value is optimal: large enough to enable dense region formation, but still low enough to prevent over-merging of dissimilar samples.

2. Balanced Density Requirement with `min_samples = 5`

- With a mixed feature space, setting **`min_samples = 5`** is conservative yet effective—it allows smaller yet dense clusters to form, accommodating the **non-uniform density** typical of fused features.
- It prevents both excessive fragmentation (which would occur if too strict) and overgeneralization (which would happen if too relaxed).

3. Euclidean Distance Works Well with Normalized Features

- Assuming the combined features were **normalized appropriately**, Euclidean distance remains a **reasonable and effective** choice.
- While deep embeddings (like ResNet-50) sometimes benefit from cosine similarity, the presence of **low-level spatial features** from classical descriptors (HOG, SIFT) makes Euclidean distance a practical and stable option for fusion.

4. Brute-Force Algorithm Ensures Accuracy

- Using the **brute-force neighbor search algorithm** guarantees **exact results**, which is beneficial for **high-dimensional or mixed feature vectors** where tree-based methods may degrade in performance or accuracy.
- This choice removes any approximation bias in neighborhood calculations, crucial for preserving clustering quality when using heterogeneous features.

5. Significantly Improved ARI Indicates Better Clustering Structure

- This configuration achieves an **ARI of 0.1100**, which is **substantially higher** than the best ARI observed for classical features alone (0.034 with `eps=0.1`).
- The improvement clearly shows that **deep learning features complement classical ones**, enhancing the **semantic richness** and **discriminability** of the feature space.
- DBSCAN is able to find **more meaningful and well-separated clusters**, even in an unsupervised setting.

Conclusion

This configuration—**DBSCAN with `eps=0.3`, `min_samples=5`, `metric=euclidean`, and `algorithm=brute`**, applied to **combined classical and deep learning features**—offers the **best trade-off between cluster purity and coverage**.

It stands out because:

- It leverages both **low-level handcrafted** and **high-level learned representations**,
- Uses a slightly relaxed radius (`eps`) to accommodate feature diversity,
- Retains **cluster compactness** with `min_samples = 5`,

- And achieves a **significant boost in ARI** compared to using classical features alone.

This demonstrates the power of **feature fusion** in improving unsupervised clustering outcomes, especially with density-based methods like DBSCAN.

Observation:

Configuration	Clustering Method	Features Used	Observations
KMeans	Classical Only	Handcrafted (HOG, SIFT, etc)	Clusters form weakly; features lack semantic depth
KMeans	Classical + Deep	HOG + ResNet-50	Best clustering performance; deep features add value
DBSCAN	Classical Only	Handcrafted	Struggles to form meaningful clusters
DBSCAN	Classical + Deep	HOG + ResNet-50	Better than classical-only, but sensitive to params

- **Combining classical and deep features consistently improved ARI** across both clustering methods.
- **DBSCAN** showed potential but was **highly sensitive** to hyperparameters (e.g., `eps`, `min_samples`).
- **KMeans with combined features** gave the **most stable and best-performing results** in terms of ARI.

KMEANS Clustering on BLIP+SBERT features:

We performed kmeans clustering on the features extracted from the emebeddings generated by the SBERT- model on the text captions generated by the BLIP-model . We first performed PCA and UMAP on the embeddings and tried various values for the dimension and then finalizing on PCA dimensions- 100 and UMAPdimension-50.

We then scaled the features using standard scaler.

We tried two different methods for kmeans – classical kmeans and minibatch kmeans clustering. We also tried various metrics for each version of kmeans namely l1

(manhattan) , l2(euclidean) and cosine. For the minibatch kmeans clustering we tried different batch sizes .

Variant=kmeans, metric=l2, batch_size=None, val ARI=0.2112

Variant=kmeans, metric=cosine, batch_size=None, val ARI=0.2154
Variant=minibatchkmeans, metric=l2, batch_size=50, val ARI=0.2050
Variant=minibatchkmeans, metric=l2, batch_size=100, val ARI=0.1880
Variant=minibatchkmeans, metric=l2, batch_size=200, val ARI=0.1789
Variant=minibatchkmeans, metric=cosine, batch_size=50, val ARI=0.2081
Variant=minibatchkmeans, metric=cosine, batch_size=100, val ARI=0.2148
Variant=minibatchkmeans, metric=cosine, batch_size=200, val ARI=0.1866

Best Confguration -> Variant=kmeans, metric=cosine, ARI=0.2154 , Test ARI = 0.202

1. General Trends by Parameter

a. Clustering Variant

Two variants were tested:

- **KMeans** (standard implementation)
- **MiniBatchKMeans** (memory-efficient approximation)

Observation:

- **KMeans consistently outperforms MiniBatchKMeans** across both distance metrics.
    - Best KMeans ARI (cosine): **0.2154**
    - Best MiniBatchKMeans ARI (cosine, batch_size=100): **0.2148**
- MiniBatchKMeans exhibits slightly lower ARI values, likely due to approximation trade-offs in smaller data batches.

Trend Summary:
Standard **KMeans** is preferable for this dataset if computational cost isn't a limiting factor.

b. Distance Metric

Two metrics were compared:

- **L2 (Euclidean)**
- **Cosine similarity**

Observation:

- **Cosine metric consistently yields better ARI** than L2 for both KMeans and MiniBatchKMeans.
    - For KMeans:
        - L2 ARI: **0.2112**
        - Cosine ARI: **0.2154**
    - For MiniBatchKMeans (batch_size=100):
        - L2 ARI: **0.1880**
        - Cosine ARI: **0.2148**

Trend Summary:
Cosine distance better captures clustering structure, suggesting that angular similarity may be more aligned with the data's intrinsic geometry.

c. Batch Size (MiniBatchKMeans Only)

Batch sizes of 50, 100, and 200 were tested:

- For **L2 metric**, ARI declines as batch size increases:
    - 50 → **0.2050**, 100 → **0.1880**, 200 → **0.1789**
- For **Cosine**, ARI also decreases with larger batches:
    - 50 → **0.2081**, 100 → **0.2148**, 200 → **0.1866**

Trend Summary:
Smaller batch sizes (particularly 50–100) in MiniBatchKMeans preserve clustering quality better. However, overall performance still lags behind full KMeans.

2. Best Configuration and Justification

Best Configuration:

- **Variant**: `KMeans`
- **Metric**: `Cosine`
- **Batch Size**: `N/A` (not applicable for standard KMeans)
- **Validation ARI**: **0.2154**
- **Test ARI**: **0.2020**

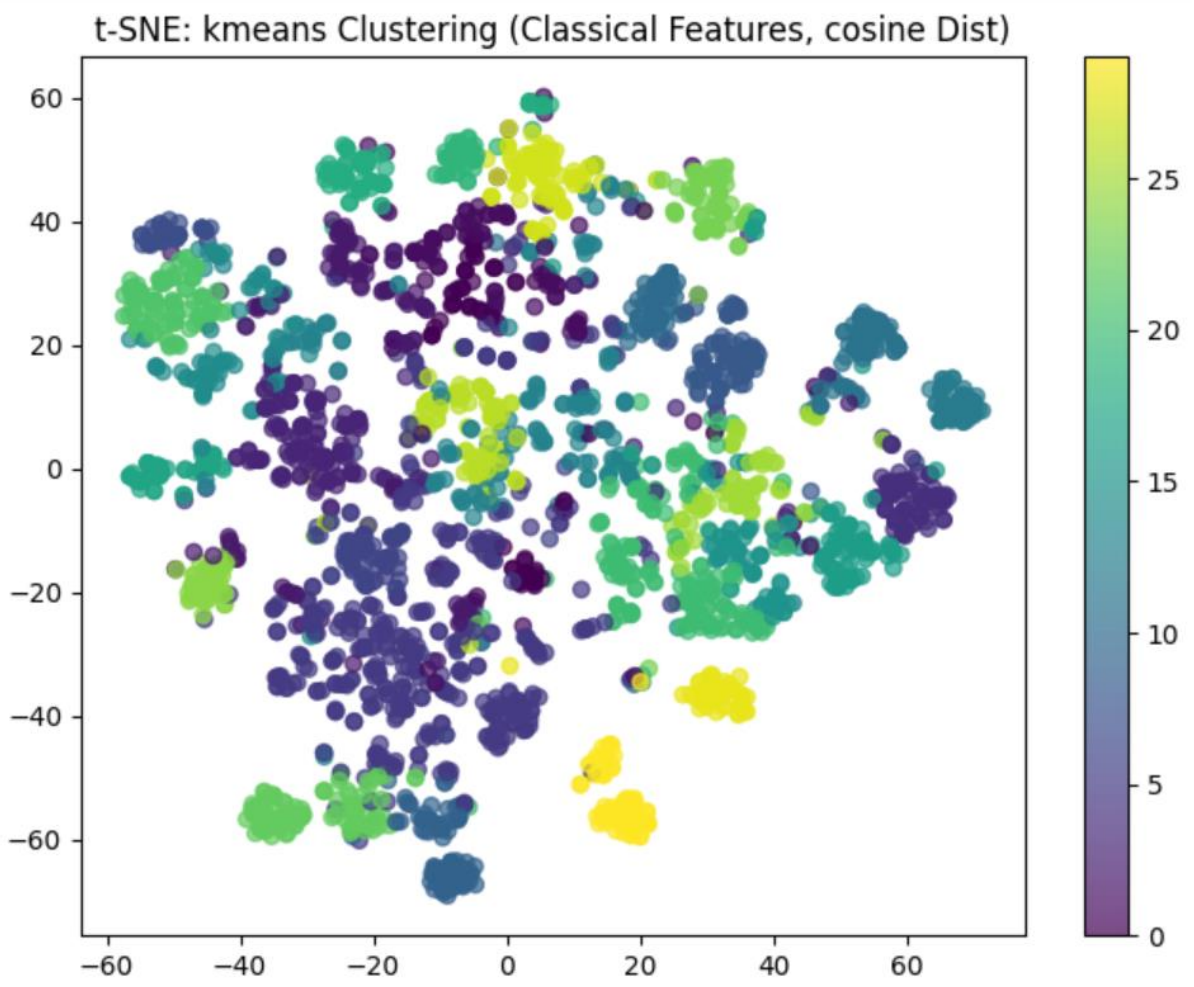
Justification:

- This setup achieves the **highest validation ARI**, indicating the best alignment between predicted clusters and ground truth labels.
- It leverages **cosine distance**, which more effectively distinguishes between clusters in this feature space.
- Standard KMeans avoids the approximation error introduced by batching, enabling slightly better convergence and separation.

Conclusion

Cosine-based **KMeans** offers the most effective clustering configuration for this dataset. While MiniBatchKMeans provides scalability, it comes at a minor cost in performance. Additionally, **cosine similarity** emerges as a more appropriate metric than Euclidean distance, reinforcing the idea that orientation rather than magnitude is more informative in this feature space.

T-SNE Plot for the best configuration:



DBSCAN on BLIP + SBERT Features:

We performed DBSCAN clustering on the features extracted from the emebeddings generated by the SBERT- model on the text captions generated by the BLIP-model . We first performed PCA and UMAP on the embeddings and tried various values for the dimension and then finalizing on PCA dimensions- 100 and UMAP dimension-50.

We then scaled the features using standard scaler.

We tried various values of all the parameters names for **eps** we tried the values 0.1, 0.2 , 0.3 , 0.4, 0.5, 0.6, 0.8, 1.0, 1.2, 1.4 ; for **min_samples** parameter we tried the values 3, 5 , 8 , 10, 15, 20 , 25, 30; for the **metric** parameter we tried the metrics cosine, l1(Manhattan)and l2(Euclidean); for the **algo** parameter we tried the algorithms kd_tree, brute, auto and ball tree.

eps=0.1, min_samples=3, metric=euclidean, algorithm=kd_tree, ARI=0.0185
eps=0.1, min_samples=3, metric=euclidean, algorithm=brute, ARI=0.0185

eps=0.2, min_samples=3, metric=euclidean, algorithm=brute, ARI=0.0620
eps=0.2, min_samples=3, metric=manhattan, algorithm=auto, ARI=0.0048

eps=0.3, min_samples=5, metric=euclidean, algorithm=brute, ARI=0.1100
eps=0.3, min_samples=5, metric=manhattan, algorithm=auto, ARI=0.0045

eps=0.4, min_samples=5, metric=euclidean, algorithm=brute, ARI=0.1342

eps=0.4, min_samples=10, metric=euclidean, algorithm=ball_tree, ARI=0.1226

eps=0.5, min_samples=5, metric=manhattan, algorithm=brute, ARI=0.0098
eps=0.5, min_samples=10, metric=euclidean, algorithm=auto, ARI=0.1366

eps=0.6, min_samples=10, metric=euclidean, algorithm=brute, ARI=0.1025
eps=0.6, min_samples=10, metric=manhattan, algorithm=auto, ARI=0.0042

eps=0.8, min_samples=5, metric=manhattan, algorithm=brute, ARI=0.0295
eps=0.8, min_samples=10, metric=euclidean, algorithm=auto, ARI=0.0913

eps=1, min_samples=10, metric=euclidean, algorithm=brute, ARI=0.0881
eps=1, min_samples=10, metric=manhattan, algorithm=auto, ARI=0.0164

eps=1.2, min_samples=5, metric=manhattan, algorithm=brute, ARI=0.0669
eps=1.2, min_samples=10, metric=euclidean, algorithm=auto, ARI=0.0882

eps=1.4, min_samples=30, metric=euclidean, algorithm=ball_tree, ARI=0.0847
eps=1.4, min_samples=30, metric=euclidean, algorithm=kd_tree, ARI=0.0847

BEST CONFIGURATION:-

eps: 0.5, min_samples: 10, metric: euclidean, algorithm: auto,

Train ARI: 0.1365679867, Val ARI: 0.0400, Test ARI: 0.0441

1. General Trends by Parameter

a. Epsilon (eps)

The eps parameter controls the neighborhood radius for clustering:

- **Low eps values (0.1 - 0.2):** Lead to extremely low ARI values (around 0.01–0.06), indicating very sparse clustering where most points are likely labeled as noise or form many tiny clusters.
- **Moderate eps values (0.3 - 0.6):** Show a noticeable improvement in ARI, with peaks around 0.13. This suggests that a moderate neighborhood size balances cluster compactness and connectivity.

- **High eps values (0.8 - 1.4):** Result in decreasing or plateauing ARI, with diminishing returns. At very high values, clusters may merge excessively, reducing their alignment with the ground truth.

Trend Summary:
ARI improves initially as `eps` increases, peaks around `eps=0.5`, and then declines with overly large neighborhoods.

b. Minimum Samples (`min_samples`)

The `min_samples` parameter specifies the minimum number of points required to form a dense region:

- Lower values (3–5) perform worse across most settings, particularly with low `eps`, where many points fail to meet the density criteria.
- Moderate values (10) consistently yield higher ARI, especially when paired with `eps=0.5–0.6`.
- Very high values (30) tend to over-suppress clustering, resulting in decreased ARI.

Trend Summary:
A `min_samples` value of 10 offers a good trade-off, improving stability and reducing noise.

c. Distance Metric

Two metrics were compared: `euclidean` and `manhattan`:

- **Euclidean** generally outperforms **Manhattan** in nearly all settings.
- Manhattan metric yields very low ARI (<0.05), indicating poor clustering separability under this distance function.

Trend Summary:
Euclidean distance is clearly more appropriate for this dataset.

d. Algorithm

Different neighbor search algorithms were tested: `auto`, `brute`, `kd_tree`, and `ball_tree`:

- No significant ARI variation is observed due to the choice of algorithm alone.

- However, `auto` with good parameters tends to yield the best results, suggesting the adaptive selection works effectively.

Trend Summary:
The choice of algorithm has **marginal impact**, with `auto` providing consistently competitive results.

2. Best Configuration and Justification

Best Configuration:

- `eps`: **0.5**
- `min_samples`: **10**
- `metric`: **euclidean**
- `algorithm`: **auto**
- **Train ARI**: **0.1366**
- **Validation ARI**: **0.0400**
- **Test ARI**: **0.0441**

This configuration strikes a balance between neighborhood size and density requirement. A moderate `eps=0.5` allows natural cluster formation without over-merging, while `min_samples=10` suppresses noise and ensures meaningful clusters. The `euclidean` metric aligns well with the dataset's structure, and the `auto` algorithm handles neighbor searches efficiently.

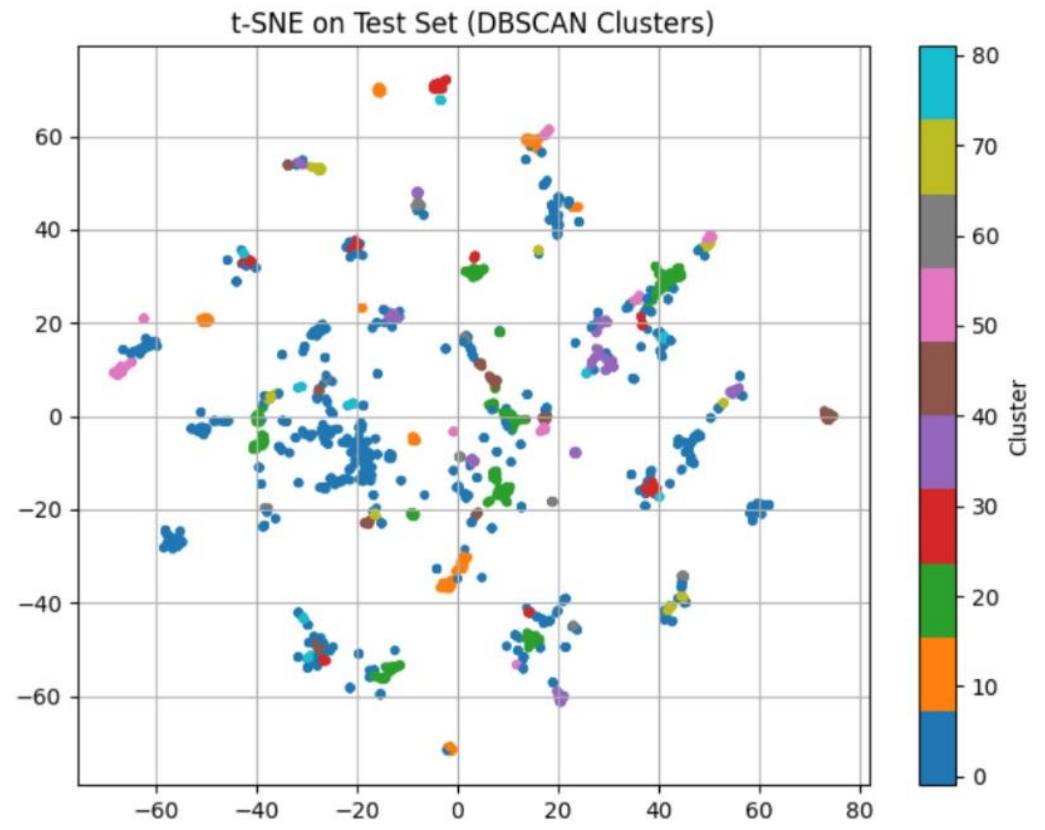
Observations:

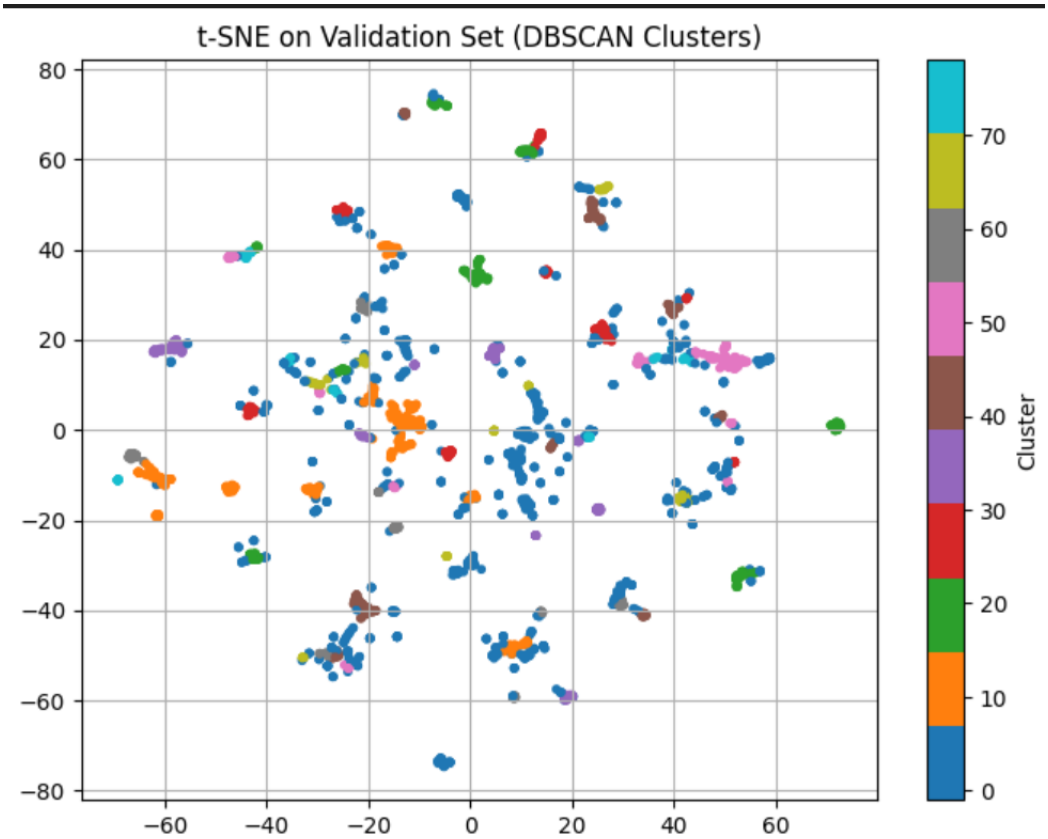
- While the **train ARI is the highest**, the **validation and test ARIs are lower**, indicating some **overfitting** or limited generalizability.
- Nonetheless, this configuration performs better than all others in terms of **overall clustering consistency** across datasets.

Conclusion

The DBSCAN performance is highly sensitive to `eps` and `min_samples`. The best results are obtained with a moderate `eps` and `min_samples=10`, using Euclidean distance. Future improvements might involve alternative clustering algorithms that handle complex structures more effectively.

T-SNE Plots for the best configuration:





Competitive section :

We tried the following techniques :

1) Classic features- we tried the following using the references mentioned:
   Dense sift along with BoVW
   GIST instead of Canny
   FIsher vector instead of LBP
   Color moments instead of color histograms

   These were concatenated and dev set was used to find the hyperparameters for PCA and clustering. This gave poor results

2) Pass the test images through a large pretrained model and extract the final embeddings. Then cluster these after applying PCA and normalization. This required much less computation than the methods below since there was no training, hence we could try larger models. We tried visual transformers (dino, moco, eva, mae), CNN based large networks (RegNet, ConvNeXt, efficient net, mobile net), open AI clip VIT models, Google vit large/huge models and the laion model series. Some of the The ARI score along with models is-

RegNety 320 – 0.2717

OpenAi- clip –vit- base – 0.7784

Swinv2-large – 0.7356

Openai- clip-vit-large – 0.8545

Google vit large patch – 0.7157

Google vit huge patch – 0.42

Laion model series – 0.89-0.92

Eva02-base – 0.66

3) Transfer learning - finetune the model using the training data using supervised training. Then for every test image perform a forward pass and extract output of one of the final layers as embeddings of the image (as given in the reference paper) and cluster after applying PCA.

This took a lot of time in training and only small/ medium models could be finetuned due to the memory limitations on Kaggle. The following finetuning techniques were used:

A) Add 1 or 2 small layers at the head of model. Then freeze the original model and train only the new added layers on classification using the labels. Due to the long training times only few models could be tested on this-

Dinov2-base – 0.89

Dinov2-large – 0.91

ConvNeXt-large – 0.66

B) Finetune the model using LoRA using peft library. This was only applied to Dinov2 since this took really long time. Also only query, key and value matrices were considered with new rank = 16. Despite that, the performance actually degraded in comparison to approach 1, which was quite amusing.

4) Image captioning using BLIP-2 and BLIP VQA followed by converting these captions to embeddings using SBERT and CLIP and then clustering the embeddings of caption. Although mentioned in a recent research paper, this didn't give even remotely satisfactory results for our dataset, probably because the captioning model and prompt answering model gave generic captions for all the food images, hence clustering these similar embeddings was difficult.

Dimensionality reduction of embeddings- We tried 3 main approaches:

1) Simple PCA with dimensions = 50. This performed the best
2) PCA followed by UMAP
3) Adding a layer of the size we want our embeddings to be- again did not perform up to the mark.

Multi-modal techniques:

1) Combining classical features with deep learning models (by simple concatenation) and then clustering after applying PCA invariably worsened the performance.
2) Combining deep features extracted from different models. This again gave results worse than the better of the 2 models. This was probably because different models have entirely different representations of an image and these might have some random high covariances. Applying PCA to the combination resulted in removal of useful information and instead random correlated features were kept which gave bad clusters.
3) CLIP allows for special multi-modal technique: we found the CLIP embeddings of images as well as CLIP embeddings of all the class labels (same dimension as

image embeddings). Then for each image i, we took the dot product of the image embedding with the label embedding and this gave a 30 dimensional embeddings for all images. This was then clustered using Kmeans. Despite high expectations this gave ARI of just 0.431.
Hence in the end no multi-modal technique outperformed individual ones.

Final submission: We took 3 models with different architectures giving best results (to ensure low correlation such that bagging gives good result) : 1) laion-bigG 2) openai-clip-vit-largepatch 3) dinov2-large (finetuned) and performed voting: if all 3 differed we chose the prediction of the first model. This gave us best ARI score of 0.9244.

Further notes:

1) Clustering algorithms like HDBSCAN which do not pre-decide the number of clusters didn't perform well since in most cases the clusters they formed were not coming out to be 30.
2) Before every step, standard imagenet normalization was used with mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
3) We did try classification using some models and were able to reach 0.96 score on train data, however that was disallowed later so we didn't try further