Artificial Intelligence for Medical Image Analysis

Pritam Kumar -24021

September 2025

1 Source of Artifacts

In each of the following images in Figure 1, (i) find the source of the artifact, and (ii) list the solutions to reduce/overcome each of them.

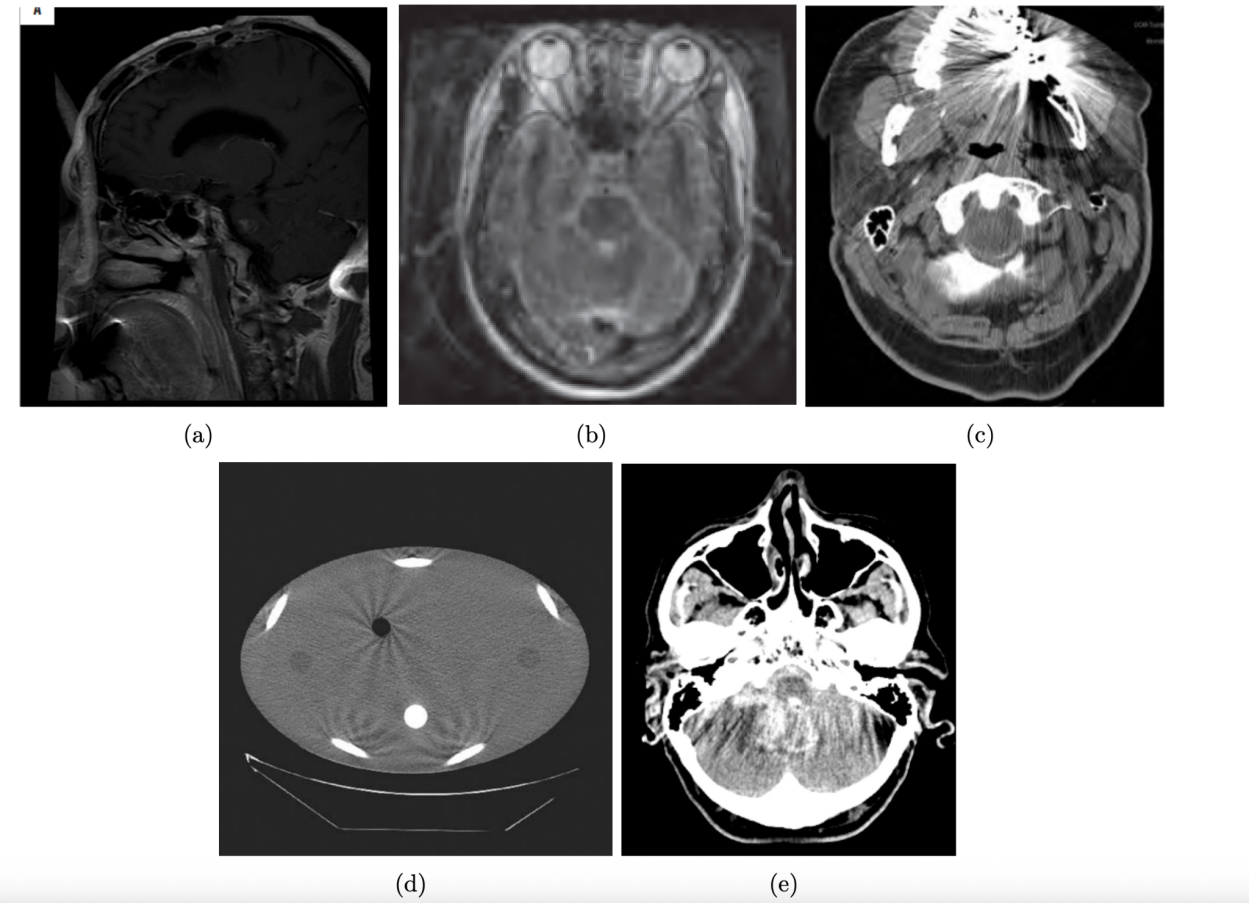

Figure 1: Artifacts

Answer:

(a) **Sagittal MRI (artifact visible near nose/air cavities)**
Source of artifact: Magnetic susceptibility artifact due to air–tissue interfaces (sinus region).
Solutions:

- Use spin-echo instead of gradient-echo sequences
- Increase bandwidth
- Shorten echo time (TE)
- Use parallel imaging or higher-order shimming

(b) **Axial MRI (blurring/ghosting across phase-encoding direction)**
Source of Artifacts: Motion artifact (Because of eye movement or patient head motion).
Solutions:

- Patient immobilization or fixation devices
- Use faster sequences (e.g., single-shot EPI, parallel imaging)
- Motion correction algorithms
- Apply saturation bands near eyes/sinuses

(c) Axial CT (bright streaks from dense structures like dental fillings)
Source of artifact: Beam hardening/streak artifacts from metal (dental implants).
Solutions:

- Use metal artifact reduction (MAR) algorithms
- Increase tube voltage (kVp)
- Use iterative reconstruction methods
- Reposition patient if possible

(d) **CT phantom (star-shaped streaks from high-density inserts)**
S**ource of artifact:** Beam hardening & photon starvation due to dense objects. **Solutions:**

- Beam hardening correction software
- Filtration of X-ray beam
- Iterative reconstruction techniques
- Reduce object density in phantom setup

(e) **Axial CT (grainy/streaky image with high contrast edges)**
Source of artifact: Partial volume averaging (when thick slices average tissues of different densities).
Solutions:

- Use thinner slices
- Employ high-resolution reconstruction
- Optimize scan angle to avoid mixing different tissues

2 Implementation of Gaussian Mixture Model(GMM):

The most important aspect of the GMM is the Gaussian Distribution. For single variable z, the Gaussian Distribtuion can be written in the form

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma)^{1/2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{1}$$

where μ is the mean and $\sigma^2$is the variance. For a D-dimensional vector x, the multivariate Gaussian distribution can be written as

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{2}$$

where $\boldsymbol{\mu}$ is a D-dimensional mean vector and $\boldsymbol{\Sigma}$ is a $D \times D$ covariance matrix. The matrix, $\boldsymbol{\Sigma}$ is a real, symmetric matrix.

Mixtures of Gaussian

Superposition of K Gaussian densities of the form

$$p(x) \quad = \quad \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k) \tag{3}$$

which is called *mixture of Gaussians.* The parameters π_k are called *mixing coefficients*

$$\sum_{k=1}^{K} \pi_k \quad = \quad 1 \tag{4}$$

We introduce a K-dimensional binary variable $\boldsymbol{z}$. $\boldsymbol{z}$ is a one-hot encoded vector in which a particular z_k is 1 and all other elements are 0.

$$p(z_k = 1) = \pi_k$$

Since $\boldsymbol{z}$ is a one-hot encoded vector, we can write the distribution in the form

$$p(\boldsymbol{z}) \quad = \quad \prod_{k=1}^{K} \pi_k^{z_k} \tag{5}$$

The conditional distribution of $\boldsymbol{x}$ given a particular value of $\boldsymbol{z}$ is a Gaussian.

$$p(x|z_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$$

Since

$$p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z}),$$

we can obtain the marginal distribution

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z}),$$

by summing over all possible states of $\boldsymbol{z}$.

$$p(\boldsymbol{x}) \quad = \quad \sum_{z} p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) \quad = \quad \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k) \tag{6}$$

The above equation is the marginal distribution of $\boldsymbol{x}$

We shall consider the conditional probability of $\boldsymbol{z}$ given $\boldsymbol{x}$. Let us denote this quanity as $\gamma(z_k)$ which is equal to $p(z_k = 1|\boldsymbol{x})$ Using Baye's theorem,

$$y(z_k) \equiv p(z_k = 1|\boldsymbol{x}) = \frac{p(z_k = 1) p(\boldsymbol{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1) p(\boldsymbol{x}|z_j = 1)} = \frac{\pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x \mid \mu_j, \Sigma_j)} \tag{7}$$

We call π_k as `prior probability` and $\gamma(z_k)$ as the `posterior probability` or `responsibility` once we have observed $\boldsymbol{x}$ The log-likelihood function is given by

$$\ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right] \tag{8}$$

To maximize the log-likelihood, we set the derivative of equation 8 with respect to μ_k equal to 0.

$$0 = \sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x \mid \mu_j, \Sigma_j)} \boldsymbol{\Sigma_k}(x_n - \boldsymbol{\mu_k}) \tag{9}$$

and we obtain

$$\boldsymbol{\mu_k} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x_n} \tag{10}$$

where,

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{11}$$

We can say that N_k is the effective number of points assigned to cluster k. Also, the mean $\boldsymbol{\mu_k}$ for the k^{th} Gaussian component is obtained by taking a weighted mean of all the points in the data set, in which the weighting factor for data point x_n is given by `posterior probability` $\gamma(z_{nk})$ that component k was responsible for generating x_n.

If we set the derivative of (8) to 0 with respect to Σ_k then we get

$$\boldsymbol{\Sigma_k} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x_n} - \boldsymbol{\mu_k})(\boldsymbol{x_n} - \boldsymbol{\mu_k})^T \tag{12}$$

And again after setting the derivative with respect to π_k, we get the following

$$\pi_k \qquad = \qquad \frac{N_k}{N} \tag{13}$$

We can interpret π_k as the *mixing coefficient* for the k^{th} component is given by the average responsibility which that component takes for explaining the data points. From the equations (10), (12), and (13), none can be obtained in closed form.

Algorithm 1 EM for Gaussian Mixtures

1: Initialize the means(From k-means) $\boldsymbol{\mu}_k$, covariances(from k-means) $\boldsymbol{\Sigma}_k$, and mixing coefficients π_k. Evaluate the initial log likelihood.

2: **E-step:** Evaluate responsibilities

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

3: **M-step:** Update parameters

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})\, \boldsymbol{x}_n$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T$$

$$\pi_k^{\text{new}} = \frac{N_k}{N}, \qquad N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

4: Evaluate log likelihood and check convergence. If not converged, return to step 2. =0

Initialization of means with K-means

Initialization: The means $\boldsymbol{\mu}_k$ are initialized using the centroids obtained from the K-means clustering algorithm. The covariances $\boldsymbol{\Sigma}_k$ are initialized to the empirical covariances of the assigned clusters, and the mixing coefficients π_k are set to equally probable. Equal initial weights. This means that initially, each Gaussian component is assumed to contribute equally to the overall mixture. This means that initially, each Gaussian component is assumed to contribute equally to the overall mixture.

Performance Metrics

1. **Dice Coefficient:** Measures the overlap between predicted mask P and ground-truth mask G. It is defined as

$$Dice(G, P) = \frac{2|G \cap P|}{|G| + |P|} \tag{14}$$

where

- $|G|$ = number of pixels in ground-truth mask
- $|P|$ = number of pixels in predicted mask
- $|G \cap P|$ = number of overlapping pixels (true positives)

Range: $0 \leq Dice \leq 1$
1 = perfect overlap
0 = no overlap

2. **Lesion-wise F1-score:** Instead of treating each pixel individually, it evaluates lesions (connected components) as objects. Each lesion in ground truth and prediction is matched if their IoU $\geq$ threshold.

$$Precision = \frac{\text{Matched Predicted Lesions}}{\text{Total Predicted Lesions}}, \qquad Recall = \frac{\text{Matched Ground Truth Lesions}}{\text{Total Ground Truth Lesions}}$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{15}$$

This metric focuses on detecting lesions as distinct objects, rather than just overlapping pixels. Missing a small lesion lowers the score, even if the Dice score is high.

3. **Lesion-wise Absolute Element Difference (AED):** Quantifies the difference in the number of lesions (objects) between ground truth and prediction.

$$\boldsymbol{AED} = |N_{GT} - N_{Pred}|$$

where

- N_{GT} = number of lesions in ground truth
- N_{Pred} = number of lesions in predicted mask

Range: Non-negative integer

- 0 = perfect match in lesion counts
- Higher values = bigger mismatch in lesion numbers

Even if Dice is high, the model may under-segment (miss lesions) or over-segment (detect spurious ones). AED highlights this issue.

Methodology

I first standardized the brain MRI slices (DWI) to make the intensity distribution consistent across images. Then, using `k-means`, I initialized means and variances of the different clusters for $K = 4$. Then I applied the Gaussian Mixture Model (GMM) clustering to segment each image into a fixed number of clusters. Since infarcts are expected to appear hyper-intense on DWI, I selected the infarct cluster by identifying the cluster with the brightest mean intensity, while also applying size constraints based on the fraction of total pixels to avoid selecting background noise or overly large regions. After cluster selection, I performed morphological postprocessing, which involved removing small objects, applying morphological closing to fill holes, and, in some cases, keeping the largest connected component. Finally, I evaluated the segmentation performance using the `Dice Coefficient` for pixel-wise overlap, the `lesion-wise F1 score` to measure how well individual lesions were matched, and the `absolute lesion count difference` to assess over- or under-detection of lesions.

Results and Analysis

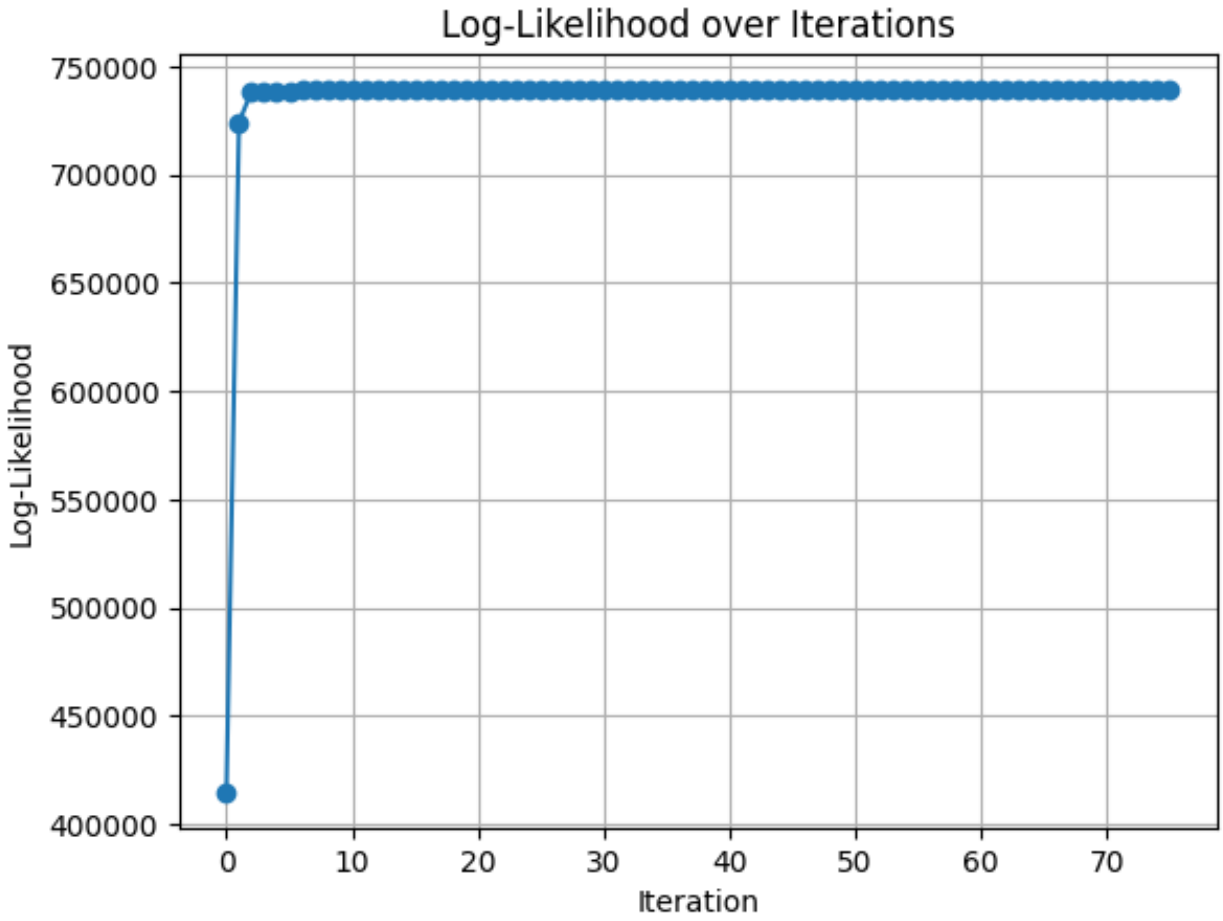

Figure 2: The plot of log-likelihood over different iterations.

Table 1: Average performance metrics on Sample and Test Images

Dataset	Average Dice	Average Lesion-wise F1	Absolute Element Difference
Sample Images	0.7697	0.7049	2.8000
Test Images	0.7646	0.7354	1.5333

The results show that the method achieved fairly consistent performance across both the sample and test datasets, with Dice scores around 0.76–0.77. This indicates that the pixel-level overlap between predicted infarct masks and ground truth is reasonably good, though not perfect. The lesion-wise F1 score is slightly lower but still acceptable, reflecting challenges in capturing individual lesions, especially when they are small or scattered. For example, in Sample Image 15, the Dice score is relatively high (0.8432), but the lesion-wise F1 is lower (0.5714) because only 2 lesions were detected out of 5 present in the ground truth, showing that smaller lesions are often missed. The average lesion count difference on the test set (1.53) confirms that there is a tendency either to under-segment or over-segment lesions. Overall, the method works well in capturing major infarcts, but improvements are needed in detecting small or multiple lesions reliably, perhaps by refining the cluster selection thresholds or enhancing postprocessing to better separate true infarcts from noise. We don't have a large number of datasets, so we can't conclude if the GMM is actually bad or not. As I noted that even if the score is not good in one image, then overall performance goes down.

3 Apply Principal Component Analysis:

We have 4708 training, 524 images for validation, and 624 test images of lungs X-rays, which are 28×28. I implemented Principal Component Analysis(PCA) from scratch(without using built-in libraries). For this task, I have followed the following pseudocode for PCA.

Algorithm 2 Principal Component Analysis (PCA)

Require: Data matrix $X \in \mathbb{R}^{n \times d}$ with n samples and d features, number of components k
Ensure: Projected data X_{pca}, selected eigenvectors, selected eigenvalues, mean vector

1: **Step 1:** Compute the mean vector:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} X_i$$

2: **Step 2:** Center the data:

$$X_{\text{centered}} = X - \mu$$

3: **Step 3:** Compute the covariance matrix:

$$C = \frac{1}{n-1} X_{\text{centered}}^T X_{\text{centered}}$$

4: **Step 4:** Compute eigenvalues and eigenvectors of C:

$$[\lambda, v] = \text{eig}(C)$$

5: **Step 5:** Sort eigenvalues in descending order and rearrange eigenvectors accordingly.
6: **Step 6:** Select the top k eigenvectors:

$$V_k = [v_1, v_2, \ldots, v_k], \quad \Lambda_k = [\lambda_1, \lambda_2, \ldots, \lambda_k]$$

7: **Step 7:** Project the data onto the selected components:

$$X_{\text{pca}} = X_{\text{centered}} V_k$$

8: **return** $X_{\text{pca}}, V_k, \Lambda_k, \mu$ =0

Figure [3] is the plot of the amount of variance explained by each principal component.

To explain 95 % of the variance, we needed *71* principal components. Further, we reduced the PCA-reduced data to 2-D data using t-SNE(t-Distributed Stochastic Neighbor Embedding). Figure [4] shows the visualization of the plot of t-SNE in two dimensions.

Working Principal of t-SNE

t-SNE is a dimensionality reduction technique that creates a low-dimensional map of high-dimensional data, making it easier to visualize and reveal patterns and clusters. It works by modeling the probability that two points are neighbors in both the high-dimensional and low-dimensional spaces, then iteratively moving points in the low-dimensional space to preserve these probabilities and cluster similar data points together.

- **Step 1: High-Dimensional Similarity-** t-SNE first calculates the probability that two points are neighbors in the original high-dimensional space.

$$p_{j|i} = \frac{e^{\frac{-\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{\frac{-\|x_i - x_k\|^2}{2\sigma_i^2}}} \tag{16}$$

- **Step 2: Low Dimensional Representation-** It then creates a similar probability distribution in

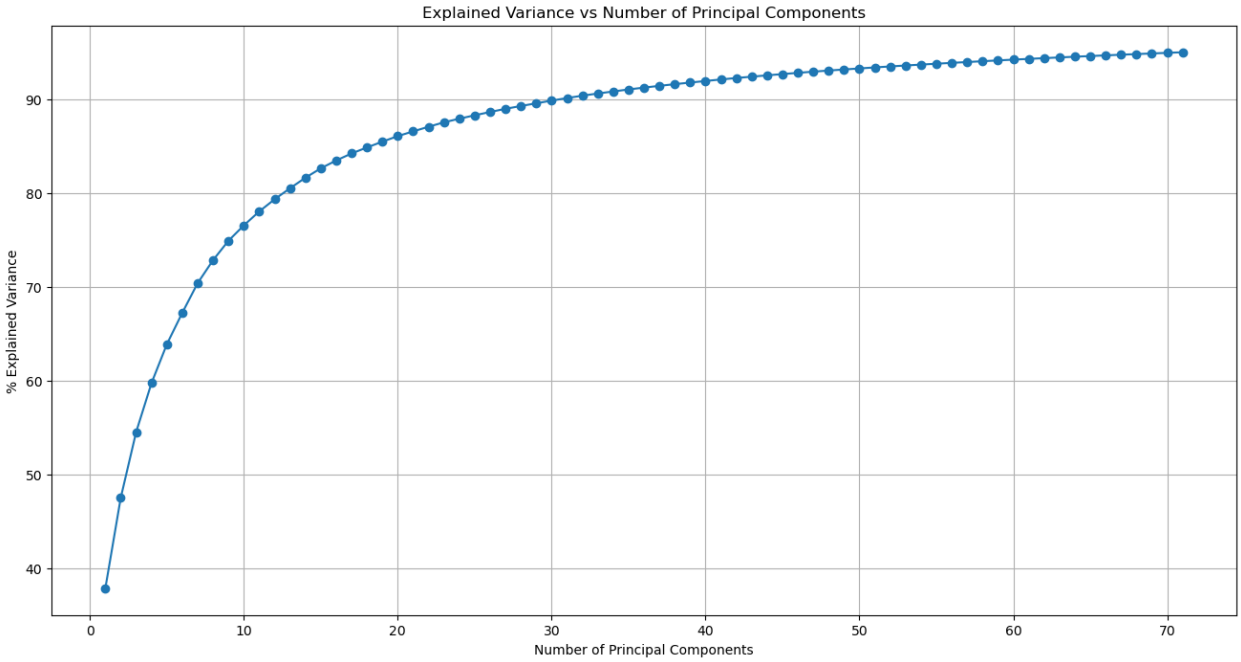

Figure 3: Percentage variance explained. The maximum variance is explained by the first component(corresponding to the largest eigenvalue of the covariance matrix.

the lower-dimensional space.

$$q_{j|i} = \frac{(1 + ||y_i - y_j||)^{-}1}{\sum_{k \neq l}(1 + ||y_k - y_l||)^{-}1} \tag{17}$$

- **Step 3: Optimization-** The algorithm iteratively moves points in the low-dimensional space, aiming to minimize the difference between these two distributions, thereby grouping similar points and separating dissimilar ones.
  t-SNE minimizes the Kullback–Leibler (KL) divergence between the two distributions $P = \{p_{j|i}\}$ and $Q = \{q_{j|i}\}$:

$$C = KL(P \,||\, Q) = \sum_{i \neq j} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}. \tag{18}$$

  The gradient of the cost function with respect to a low-dimensional point y_i is:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{j|i} - q_{j|i})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}. \tag{19}$$

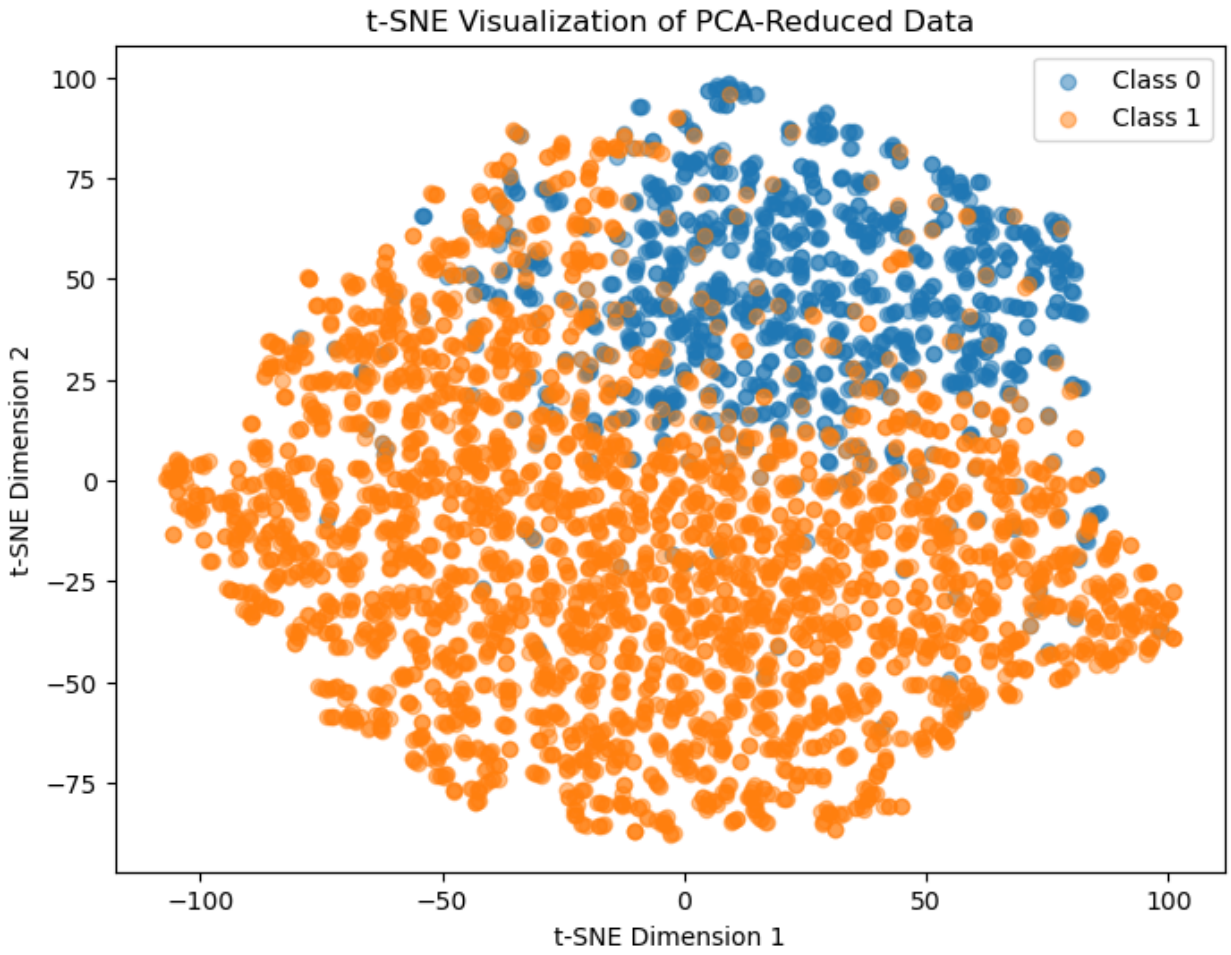

Figure 4: Projection of the high dimensional image(71) to 2 dimension.

We now have a dataset with 71 features. So, we shall need a weight vector w of dimension 71 and a bias term b

Algorithm 3 Logistic Regression Training

Require: Data matrix $X \in \mathbb{R}^{n \times d}$, labels $y \in \{0,1\}^n$, learning rate α, number of iterations T
Ensure: Trained weights W, bias b

1: Initialize weights $W \in \mathbb{R}^{d \times 1}$ with small random values
2: Initialize bias $b = 0$
3: **for** $t = 1$ to T **do**
4: **Forward Propagation:**

$$Z = XW + b, \quad A = \sigma(Z) = \frac{1}{1 + e^{-Z}}$$

5: **Compute Loss:**

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(A_i) + (1 - y_i) \log(1 - A_i) \right]$$

6: **Compute Gradients:**

$$dW = \frac{1}{n} X^T (A - y), \quad db = \frac{1}{n} \sum_{i=1}^{n} (A_i - y_i)$$

7: **Update Parameters:**

$$W := W - \alpha dW, \quad b := b - \alpha db$$

8: **end for**
9: **return** W, b =0

Algorithm 4 Logistic Regression Prediction

Require: New data X_{test}, trained parameters W, b
Ensure: Predicted labels $\hat{y}$

1: Compute $Z = X_{\text{test}} W + b$
2: Compute $A = \sigma(Z)$
3: Assign $\hat{y}_i = 1$ if $A_i \geq$ *threshold*, else $\hat{y}_i = 0$
4: **return** $\hat{y}$ =0

Results and Analysis

I trained the model `logistic_regression` using the parameters:

- Learning Rate = 10^{-5}
- Number of Epochs = 1600
- Threshold = 0.336
- Regularization coefficient $\lambda = 10^{-5}$

These parameters were obtained by hyperparameter tuning.
We have used the following formula to obtain the `Accuracy, Precision, Recall, and F1-Score`

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}, \qquad Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}, \qquad F1 - Score = \frac{2 \times precsion \times recall}{precision + recall}$$

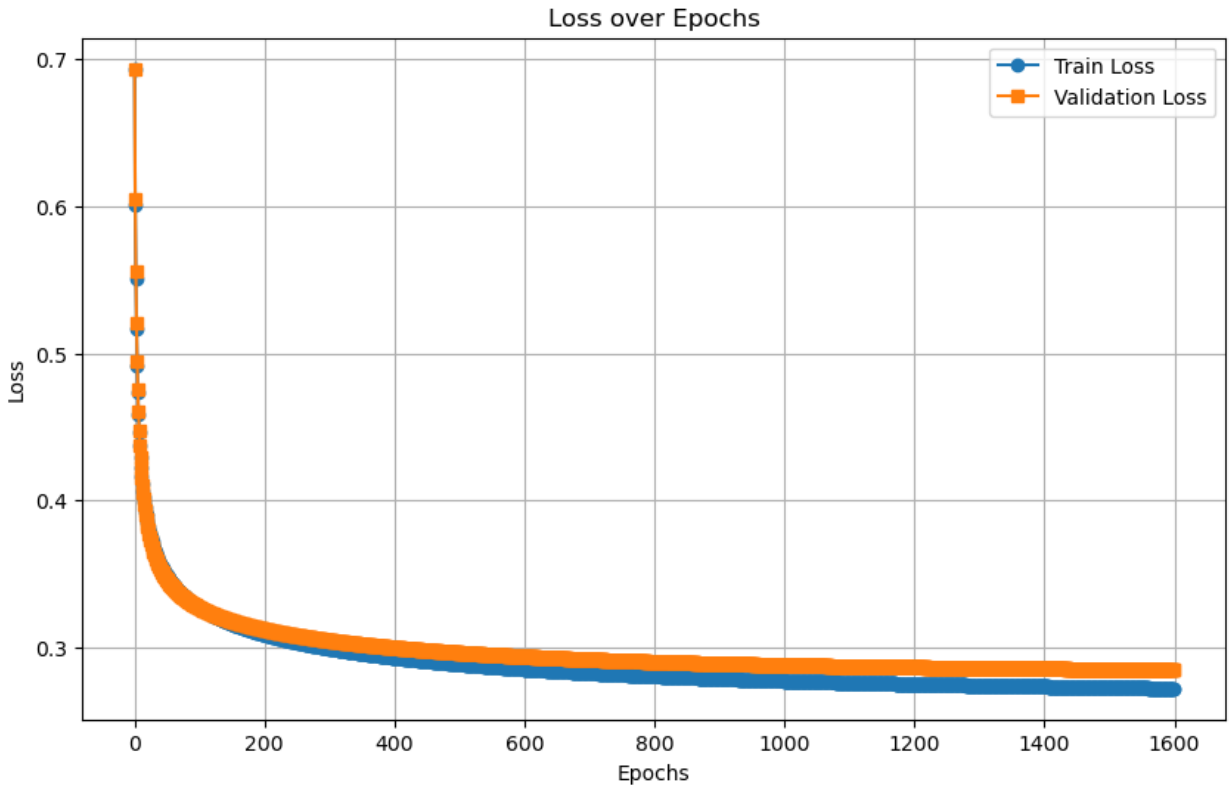

Figure 5: The graph shows the loss curve at different epochs.

We used `Binary Cross-Entropy` loss function, and the formula for the loss is given by

$$Loss = -(y * log(p) + (1 - y) * log(1 - p)) \tag{20}$$

Where y is the true label and p is the predicted probability. It measures the dissimilarity between the predicted and true probability distribution. The loss function is a strictly convex function that will have a global minimum. We can see that the curves in Figure [5] have almost become horizontal, indicating that the model has trained very well and shows no signs of overfitting. Also, validation loss is always higher than training loss. Below is the result that we obtained after training the model given in [3]

Table 2: Training and Validation Set Metrics

Dataset	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
Training Set	93.343	98.356	92.473	95.324
Validation Set	93.246	97.574	93.059	95.263

The model demonstrates strong and consistent performance across both the training and validation sets. The overall accuracy is high (93.34% for training and 93.25% for validation), indicating that the model generalizes well without significant overfitting. Precision values are slightly higher than recall (Training: 98.36% vs. 92.47%, Validation: 97.57% vs. 93.06%), suggesting that the model is conservative in predicting positive cases while still correctly identifying the majority of true positives. The F1-scores remain balanced and high (95.32% for training and 95.26% for validation), confirming that the model maintains an effective trade-off between precision and recall, and is reliable for both datasets.