CS337 Project : Reverse Image Search using LSH

Harsh Shah(200050049)
Maulinraj Parmar(200050098)
Radhika Chouhan(200050114)
Rounak Dalmia(200050119)

August 18, 2023

Contents

1 Introduction

1.1 Motivation

The motivation behind this project is from recent advancements in CSAM(Child Sexual Abuse Material) detection. *Apple* company claimed to run NeuralHash algorithm on apple devices to detect abusive materials(images) which match the images in their database. The algorithm passes the images through a neural network(robust to minor transformations to the images) to reduce the images to a low dimension vector representation. Even in this reduced dimension representation, search over all images for a single query image take lot of computational time and resources. This is resolved by using hyperplane locality sensitive hashing, which maps similar vectors to hashes which are similar as well(unlike hashes like SHA which entirely change the hash).

1.2 Aim

Through this project we aim study this algorithm and compare search using LSH and KNN(k-nearest neighbours).
We also performed adversarial attack on the trained network(ResNet34). The attack was successful, and the model started giving wrong predictions with just *0.01(max)* deviations to the pixel intensities. For more details, refer to experiment section.

2 Reverse Image Search

Reverse Image Search is a query technique used for various purposes today all that we have to do is provide an image to the system and it will help us find duplicate or similar images, instead of giving the system the related keywords or terms which might result in incorrect result, hence it is quite effective. One of the approaches for this search is to first reduce the image to a latent space of reduced dimension(using neural network) and then for any query image, search over all images in the latent space.

3 Transfer learning for embedding

We have used transfer learning technique for effective convergence to local minima of the architecture chosen on new dataset. Below are details for the same.

3.1 Pre-Processing and Embedding in latent space

To generate embeddings of all the images of dataset, we used a ResNet34 neural network. The network is pre-trained on large Imagenet dataset and then fine-tuned on Caltech-101 dataset(having 101 classes). Details of the fine-tuning procedure

1. Architecture : ResNet34
2. Pre-trained : ImageNet
3. Optimizer : Adam(initial learning rate=1e-4)
4. Loss : Cross Entropy Loss

5. Max Epochs = 10

The images(both train and test) are normalized before training(fine-tuning) the network. Finally, to generate the image embeddings in the latent space, the classification head of the fine-tuned network is removed to get embeddings of dimension **512**.

Figure 1: Caltect101 Dataset

3.2 Visualizing the embeddings

In order to analyze whether the learnt embeddings actually represent similarity between images, we have used PCA and t-sne, for analysis of clusters generated in low dimensional space.

3.2.1 PCA

PCA is an unsupervised linear dimensionality reduction and data visualization technique for very high dimensional data. As having high dimensional data is very hard to gain insights from adding to that, it is very computationally intensive. The main idea behind this technique is to reduce the dimensionality of data that is highly correlated by transforming the original set of vectors to a new set which is known as Principal component.

3.2.2 t-SNE

PCA tries to place dissimilar data points far apart in a lower dimension representation. But in order to represent high dimension data on low dimension, non-linear manifold, it is important that similar datapoints must be represented close together, which is not what PCA does. This is done efficiently by t-SNE. So, it can efficiently capture the structure of trickier manifolds in the dataset. Hence, for better analysis of the embeddings, we have also analysed the projections using t-SNE.

4 Image retrieval

4.1 K-Nearest Neighbours

We have used KNN algorithm as a baseline for comparing the improvement of search results with locality sensitive hashing. The algorithm uses the saved feature vectors of the images in database as trainset and for any query image feature, the algoorithm returns the k nearest neighbours of the query image in the latent space.

Figure 2: KNN Implementation

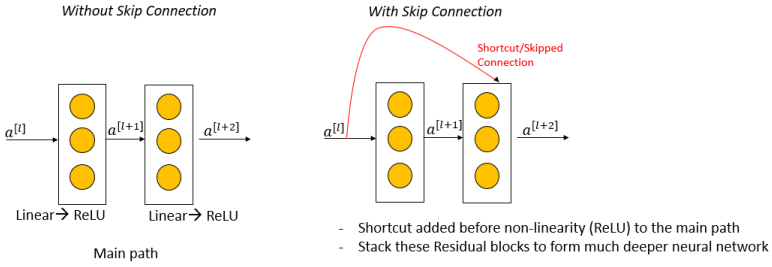

Figure 3: Residual block creation

4.2 Locality Sensitive Hashing(LSH)

Locality sensitive hashing is a a hashing technique which produces similar hashes for similar inputs to the hash algorithm. This technique is often used to reduce time complexity for searching nearest points of an query data. The algorithm produces a fixed size bit hash of a high dimensional vector(which are image features here). There are many ways for implementing LSH, inlcuding shingling, minhashing, random projections, etc. We have use random hyperplane projection LSH for enhancing the image search speed. For mathematical details of LSH, kindly refer to the appendix.

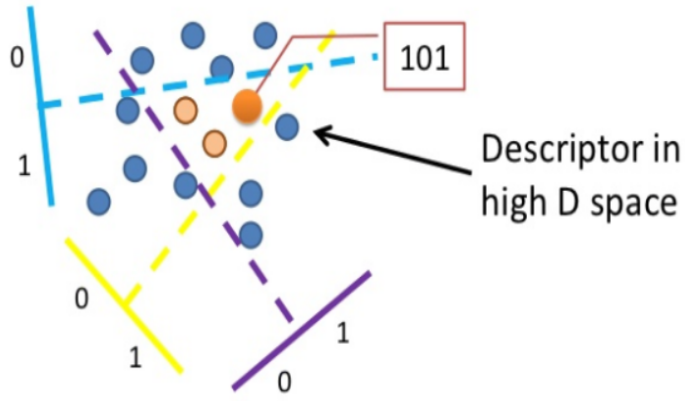

Figure 4: Hashing using projection on random hyperplanes

4.2.1 Indexing

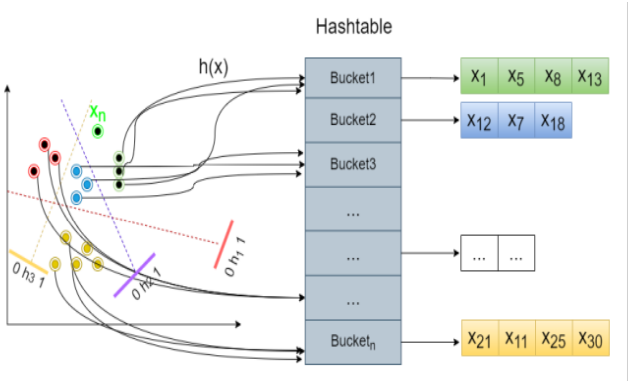

Figure 5: Indexing the hashed values into the hash table

4.2.2 Querying or Searching

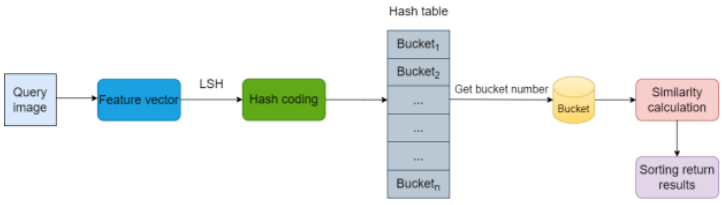

Figure 6: Hashing using projection on random hyperplanes

The query image is first input into the network, and the feature vector is produced after the feature extraction model, and then the feature vector is encoded and hashed into the homologous hash bucket of the hash table employing the LSH algorithm random projection, and the corresponding bucket number is obtained. Then, the corresponding

data in the bucket number is taken out, the distance between the query image to be retrieved and these data is calculated, and the most similar n data are returned.

4.2.3 Implementation details

For the project, we used '*lshashpy3*' package for an approximate nearest neighbor search. We used LSHash() function to index the embedded image vectors into the hash-table. To query for similar images given an image we have used lsh.query() method to get the top 'n' most similar images to the one given. lsh.query(*query_point*, *num_results*=None, *distance_func*="euclidean"): lsh.query() method takes three inputs 1.) Image vector whose similar images we want to find 2.) Number of similar images we want to find 3.) Distance metric to measure similarity.

5 Experiments and Results

Accuracy after transfer learning on Caltech-101 : 96%.
Here are the results of time complexity analysis on using 2 algorithms for feature search, KNN and LSH. Here 5 represents the time taken by both the algorithms with varying size of database. On the other hand, 5.

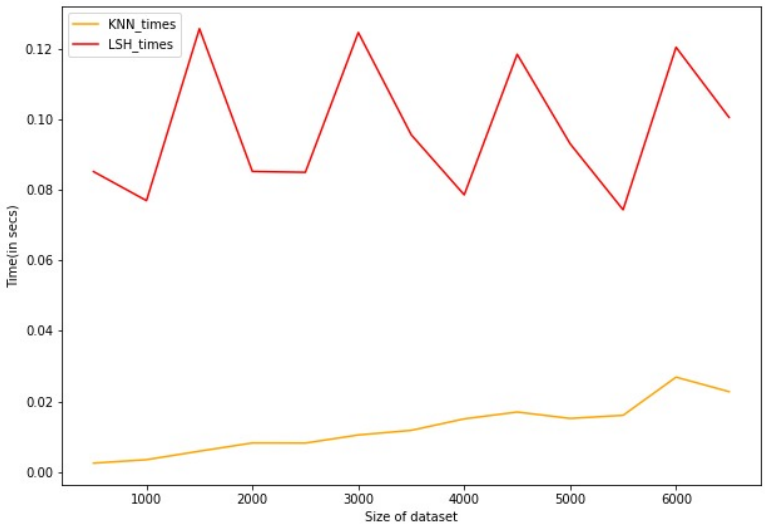

Figure 7: KNN v/s LSH(on Caltech-101 dataset)

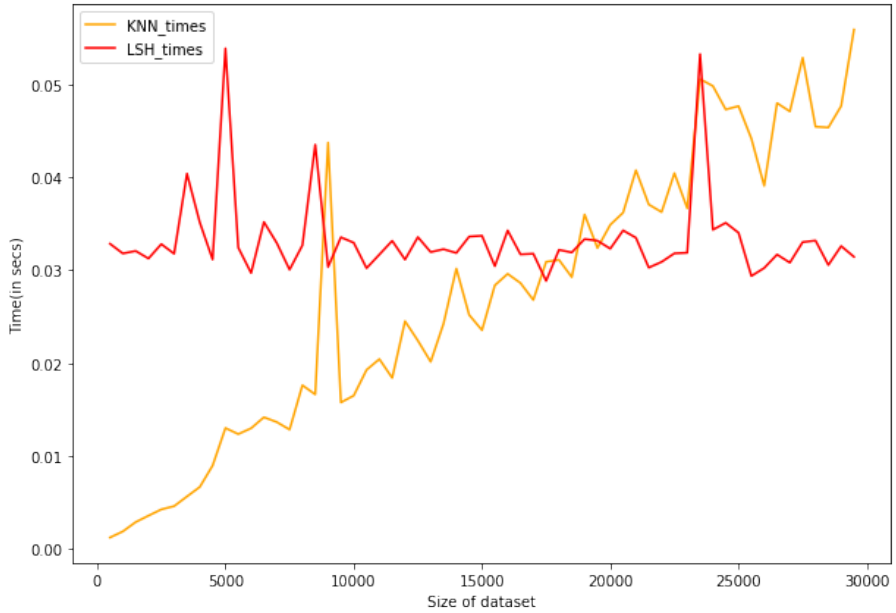

Figure 8: KNN v/s LSH(on randomly generated dataset)

As we can see for a large dataset LSH performs better that brute force KNN, but for smaller datasets KNN performs better due to vectorization in implementation of KNN algorithm while in LSH it takes more time to index the hashvalues in hashtable and lookup for similar images for smaller datasets takes more time(function calls, accessing the hash-table, etc.).

5.1 Adversarial Attacks

After open-sourcing their(*Apple's*) NeuralHash model, many advesarial attacks were made on the model to generate similar hashes for entirely different images. In order to see how robust our model is, we have carried out adversarial attack on our model using **Fast Gradient Sign Method**. Surprising, with perturbations invisible to human eyes(¡0.01 in pixel intensities), the model gave false predictions(**20%**) for many images, some of which are below.

6 Conclusion

For our experiments, we concluded that using LSH can get significant performance gain over KNN search. LSH on average gives constant time lookups with respect the database size, whereas KNN gives O(N) search time complexity(N being the size of database in terms of number of images). Further, models(especially those which are open-sourced) are very prone to white-box adversarial attacks. Minor tuned perturbations can lead to significant losses in accuracy of models, further leading to undsirable hash collisions.

Figure 9: Left : Adversarial image detected as hawksbill; Right : Crocodile

Figure 10: Left : Adversarial image detected as Pizza; Right : Gramophone

References

[1] Image Retrieval Algorithm Based on Locality-Sensitive Hash Using Convolutional Neural Network and Attention Mechanism, *Youmeng Luo, Wei Li , Xiaoyu Ma and Kaiqiang Zhang*

[2] https://towardsdatascience.com/finding-similar-images-using-deep-learning-and-locality-sensitive-hashing-9528afee02f5

7 Appendix

7.1 LSH

There are many common similarity search algorithms in the image search part. The first thought is linear search. Firstly, the similarity measure is defined, and then the similarity is calculated in pairs, and the top-n is calculated for filtering. Under the background demand of large-scale image retrieval, this time complexity is too large. At the same time, for high-dimensional sparse data, the calculation of similarity is very time-consuming. We need some approximation algorithms to improve the retrieval speed. In this study, the LSH algorithm is used to represent the original image with a low latitude(space) binary hash code to avoid directly storing the high latitude image features.

LSH constructs a hash function so that the closer the feature points in the original feature space are to be mapped by this hash function, the more likely they will fall into the same hash bucket, and vice versa. Then, when querying the image, it is only necessary to obtain the bucket number, and then take out all the data in the bucket from the corresponding bucket, perform linear matching on them, and finally find the similar images that meet the query requirements. In other words, a super large original data set is divided into several subsets after the mapping transformation of the hash function, so as to reduce the query scope to improve retrieval performance.

(1) Similarity measure and LSH hash function
(a) LSH hash function family :
It is known that $d(x, y)$ is the distance between two points x, y, h is the hash function, $h(x)$ and $h(y)$ are the hash transformation of points x, y, and $P_1 > P_2$. For any two points x, y in the high dimensional space :
If $d(x, y) \leq h_1$, there must be $Pr[(x) = h(y)] \geq P_1$.
If $d(x.y) \geq h_2$, there must be $Pr[(x) = h(y)] \leq P_2$.
In similarity calculation, different methods can be adopted to measure the similarity between two points. For dissimilar measurement methods, the hash function used in the local hash is different. There are many methods to measure the similarity of two vectors(mainly **min-hashing** and **random projections**). We used random planes projection technique to generate the hashing of a particular image embedding. Following are the steps on how LSH converts an embedding in a hash of size K :

1. Generate K random hyperplanes in the embedding dimension(that is, dimension of our embedding vector of each image)
2. Check if particular embedding is above or below the hyperplane and assign 1 or 0 accordingly(1 if above, else 0)
3. Do step 2 for each K hyperplanes to arrive at the hash value (a K-bit hash value)

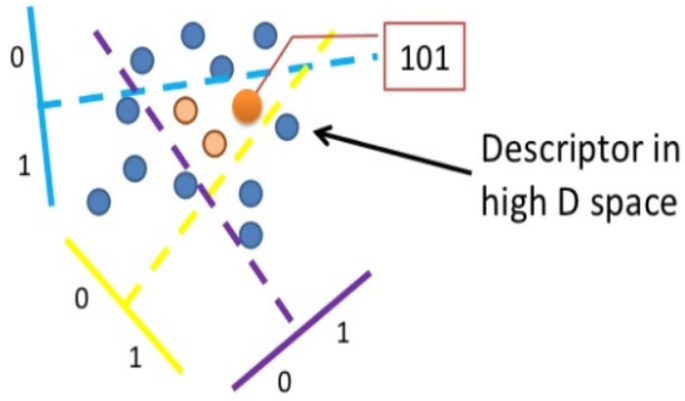

Figure 11: Hashing using projection on random hyperplanes

(b) Similarity measure using different distance metrics :
Almost similar eigenvectors can be found by representing them in k-dimensional space based on cosine distance. We use the angle between two vectors to measure whether two vectors are similar. The smaller the angle between two vectors, the more similar they are. The calculation formula of vector included angle is as follows:

$$cos(\theta) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

$$cos(\theta) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Similarly euclidean distance could be measured as $d_{euclidean}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ and L1-norm as $d_{L1-norm}(x, y) = \sum_{i=1}^{n} |x_i - y_i|$.

(2) Building Index

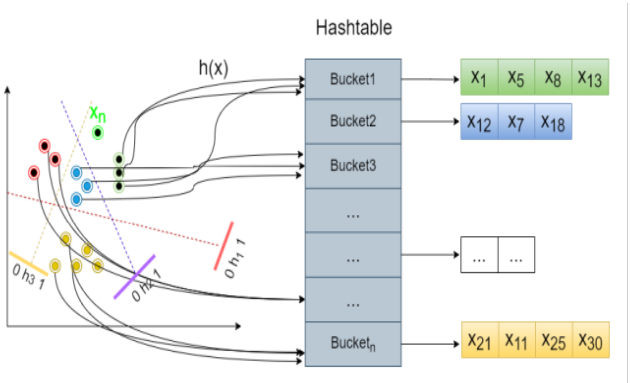

Figure 12: Indexing the hashed values into the hash table

(a) The step to building index :

1. Calculate a hash function $h(x)$ to store similar points in the same bucket.
2. For a new query point $x_n \rightarrow h(x_n) \rightarrow Bucket_n$, calculate that x_n should belong to a certain slot.

The feature vectors are projected into the m-dimensional space using the random projection method, where $m.n$, while maintaining the same cosine distance. Firstly, this paper divides the original data space with a random hyperplane. After each data is projected, it will fall into one side of the hyperplane. After multiple random hyperplane partitions, the original space is divided into many cells, and the data in each cell is considered likely to be adjacent, then, hashes the data in each cell into the corresponding slot through the hash function $h(x)$ to form a hash table. In a hash table, different numbers of buckets and their corresponding feature vectors may be created.

(3) Querying or Searching

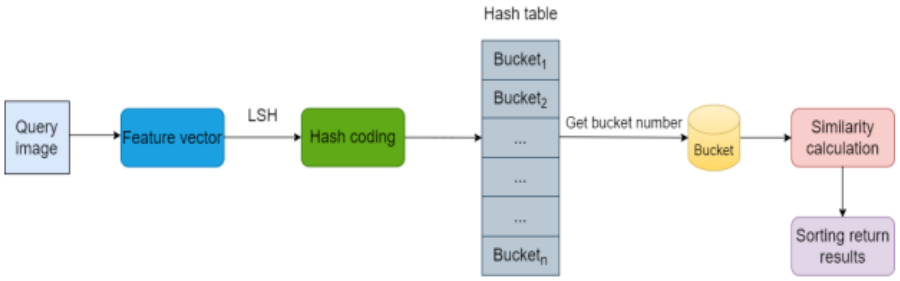

Figure 13: Hashing using projection on random hyperplanes

The query image is first input into the network, and the feature vector is produced after the feature extraction model, and then the feature vector is encoded and hashed into the homologous hash bucket of the hash table employing the LSH algorithm random projection, and the corresponding bucket number is obtained. Then, the corresponding data in the bucket number is taken out, the distance between the query image to be retrieved and these data is calculated, and the most similar n data are returned.

8 Transfer Learning

Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another. It's currently very popular in deep learning because it can train deep neural networks with comparatively little data. This is very useful in the data science field since most real-world problems typically do not have millions of labeled data points to train such complex models.

8.1 How it works

In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. In transfer learning, the early and middle layers are used and we only retrain the latter layers. It helps leverage the labeled data of the task it was initially trained on.

In transfer learning, we try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data. For example, it could be how models are composed, which allows us to more easily identify novel objects.

8.2 Why transfer learning

Transfer learning has several benefits, but the main advantages are saving training time, better performance of neural networks (in most cases), and not needing a lot of data. Usually, a lot of data is needed to train a neural network from scratch but access to that data isn't always available — this is where transfer learning comes in handy.

With transfer learning a machine learning model can be built with comparatively little training data because the model is already pre-trained. This is especially valuable in natural language processing because mostly expert knowledge is required to create large labeled data sets. Additionally, training time is reduced because it can sometimes take days or even weeks to train a deep neural network from scratch on a complex task.

9 ResNet

A residual network, or ResNet for short, is an artificial neural network that helps to build deeper neural network by utilizing skip connections or shortcuts to jump over some layers.There are different versions of ResNet, including ResNet-18, ResNet-34, ResNet-50, and so on. The numbers denote layers, although the architecture is the same.
To create a residual block, add a shortcut to the main path in the plain neural network, as shown in the figure below

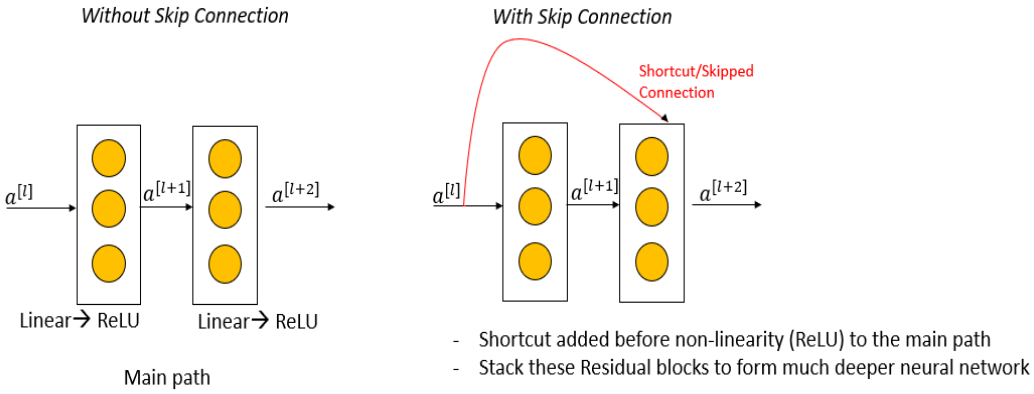

Figure 14: Residual block creation

9.1 ResNet Blocks

There are two main types of blocks used in ResNet, depending mainly on whether the input and output dimensions are the same or different.

Identity Block: When the input and output activation dimensions are the same.
Convolution Block: When the input and output activation dimensions are different from each other.

9.2 Why ResNet?

Vanishing gradient is a problem that is encountered while training artificial neural networks that involved gradient based learning and backpropagation. We know that in backpropagation, we use gradients to update the weights in a network. But sometimes what happens is that gradient becomes vanishingly small, effectively preventing the weights to change values. This leads to network to stop training as same values are propagated over and over again and no useful work is done.

To solve such problems, Residual neural networks are introduced. Also with the help pf ResNets networks with large number (even thousands) of layers can be trained easily without increasing the training error percentage.

10 Adam Optimizer

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm.

10.1 How Adam Works

Adam optimizer involves a combination of two gradient descent methodologies:

10.1.1 Momentum

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

$$w_{t+1} = w_t - \alpha w_t \tag{1}$$

where,

$$m_t = \beta m_{t-1} + (1-\beta)[\frac{\delta L}{\delta w_t}] \tag{2}$$

m_t = aggregate of gradients time t [current] (initially, $m_t = 0$)
m_{t-1} = aggregate of gradients at time t-1 [previous]
W_t = weights at time t
W_{t+1} = weights at time t+1
α_t = learning rate at time t
δL = derivative of Loss Function
δW_t = derivative of weights at time t
β = Moving average parameter

10.1.2 Root Mean Square Propagation (RMSP)

Root mean square prop or RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the 'exponential moving average'

$$w_{t+1} = w_t - \frac{\alpha_t}{(\nu_t + \epsilon)^{1/2}} * [\frac{\delta L}{\delta w_t}] \tag{3}$$

where,

$$\nu_t = \beta \nu_{t-1} + (1-\beta) * [\frac{\delta L}{\delta w_t}]^2 \tag{4}$$

W_t = weights at time t
W_{t+1} = weights at time t+1
α_t = learning rate at time t
δL = derivative of Loss Function
δW_t = derivative of weights at time t
V_t = sum of square of past gradients. [i.e sum($\delta L/\delta W_t - 1$)] (initially, Vt = 0)
β = Moving average parameter
ϵ=A small positive constant

Taking the formula used in the above two methods, we get

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)[\frac{\delta L}{\delta w_t}] \nu_t = \beta_2 \nu_{t-1} + (1-\beta_2 *)[\frac{\delta L}{\delta w_t}]^2 \tag{5}$$

Parameters Used :
1. ϵ= a small +ve constant to avoid 'division by 0' error when ($v_t -> 0$).
2. β_1 and β_2= decay rates of average of gradients in the above two methods.
3. α= Step size parameter / learning rate

11 KNN Algorithm

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.It assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories, stores all the available data and classifies a new data point based on the similarity. It can be used for Regression as well as for Classification.It is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. At the training phase it just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

11.1 Algorithm

Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance. KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

12 PCA vs T-SNE

PCA is an unsupervised linear dimensionality reduction and data visualization technique for very high dimensional data. As having high dimensional data is very hard to gain insights from adding to that, it is very computationally intensive. The main idea behind this technique is to reduce the dimensionality of data that is highly correlated by transforming the original set of vectors to a new set which is known as Principal component.

12.1 Shortcomings of PCA

1. PCA is sensitive to the scale of the features.This is why it's so important to standardize the values first.
2. PCA is not robust against outliers. Similar to the point above, the algorithm will be biased in datasets with strong outliers. This is why it is recommended to remove outliers before performing PCA.

3. PCA assumes a linear relationship between features. The algorithm is not well suited to capturing non-linear relationships. That's why it's advised to turn non-linear features or relationships between features into linear

4. Technical implementations often assume no missing values.Be sure to remove those rows and/or columns with missing values, or impute missing values with a close approximation

12.2 Why t-sne over pca?

1. Handles Non Linear Data Efficiently: PCA is a linear algorithm. It creates Principal Components which are the linear combinations of the existing features. So, it is not able to interpret complex polynomial relationships between features. So, if the relationship between the variables is nonlinear, it performs poorly. On the other hand, t-SNE works well on non-linear data. It is a very effective non-linear dimensionality reduction algorithm.

2. PCA tries to place dissimilar data points far apart in a lower dimension representation. But in order to represent high dimension data on low dimension, non-linear manifold, it is important that similar datapoints must be represented close together, which is not what PCA does. This is done efficiently by t-SNE. So, it can efficiently capture the structure of trickier manifolds in the dataset.

3. Preserves Local and Global Structure: t-SNE is capable to preserve local and global structure of the data. This means, roughly, that points which are close to one another in the high-dimensional dataset, will tend to be close to one another in the low dimension. On the other hand, PCA finds new dimensions that explain most of the variance in the data. So, it cares relatively little about local neighbors unlike t-SNE.