SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOLOFCOMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT–III –DEEPLEARNING – SCSA3015

UNIT III DIMENTIONALITY REDUCTION

Linear (PCA, LDA) and manifolds, metric learning - Auto encoders and dimensionality reduction in networks - Introduction to Convnet - Architectures – AlexNet, VGG, Inception, ResNet - Training a Convnet: weights initialization, batch normalization, hyperparameter optimization.

3.1 Components of Dimensionality Reduction

There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
    1. Filter
    2. Wrapper
    3. Embedded
- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

<u>Methods of Dimensionality Reduction</u>

The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

<u>Advantages of Dimensionality Reduction</u>

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

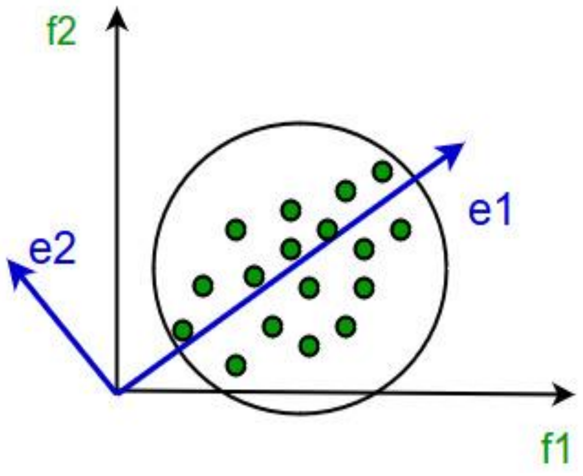
<u>Disadvantages of Dimensionality Reduction</u>

- It may lead to some amount of data loss.

- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.

3.2 Principal Component Analysis:

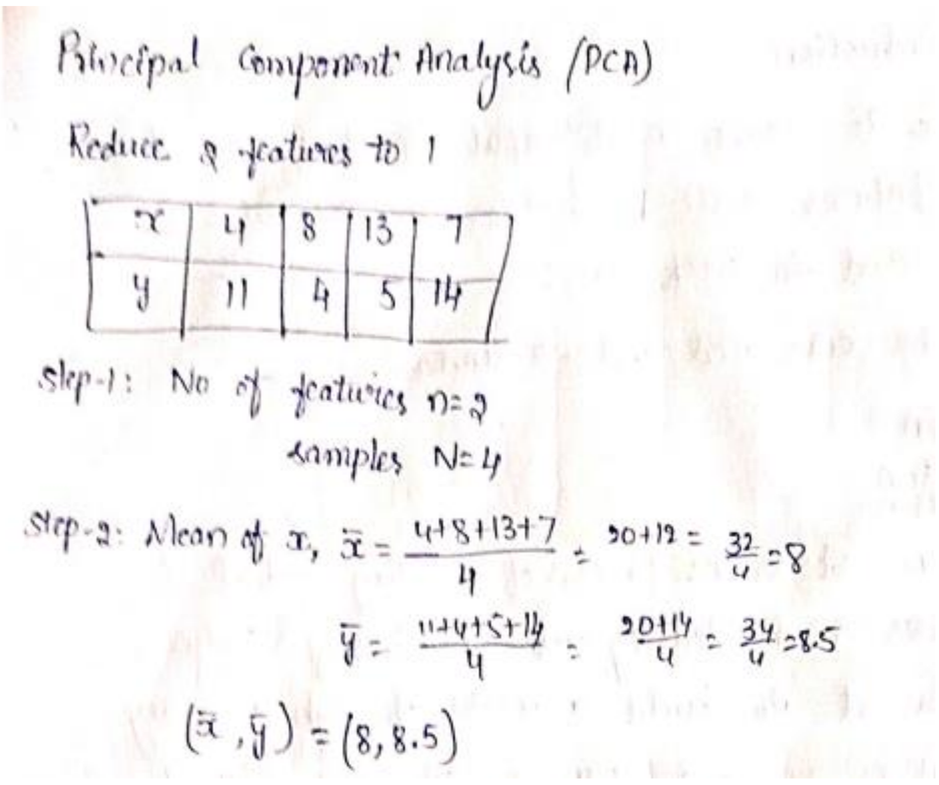
This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.



Principal Component Analysis (PCA)

Reduce 2 features to 1

x	4	8	13	7
y	11	4	5	14

step-1: No of features $n=2$

samples $N=4$

step-2: Mean of x, $\bar{x} = \frac{4+8+13+7}{4} = 20+12 = \frac{32}{4} = 8$

$\bar{y} = \frac{11+4+5+14}{4} = \frac{20+14}{4} = \frac{34}{4} = 8.5$

$(\bar{x}, \bar{y}) = (8, 8.5)$

Step-3: covariance matrix

$$\text{cov}(x,x) = \frac{1}{N-1} \Sigma (x_{ik}-\bar{x}_i)(x_{jk}-\bar{x}_j)$$

$$= \frac{1}{N-1} \Sigma (x-\bar{x})^2$$

$$= \frac{1}{4-1} [(4-8)^2+(8-8)^2+(13-8)^2+(7-8)^2]$$

$$= \frac{1}{3} [16+0+25+1] = \frac{42}{3} = 14$$

$$\text{cov}(x,y) = \text{cov}(y,x) = \frac{1}{N-1} [\overset{x-\bar{x}}{(4-8)}\overset{y-\bar{y}}{(11-8.5)} + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(4-8.5)]$$

$$= \frac{1}{3} [-4(2.5)+0+5(-3.5)+(-1)(5.5)]$$

$$= \frac{1}{3} [-10+-17.5-5.5] = \frac{1}{3}(-33) = -11$$

$$\text{cov}(y,y) = \frac{1}{N-1} \Sigma y_1$$

$$= \frac{1}{3} [(11-8.5)^2+(4-8.5)^2+(5-8.5)^2+(14-8.5)^2]$$

$$= \frac{1}{3} [(2.5)^2+ +(4.5)^2+(-3.5)^2+(5.5)^2]$$

$$= \frac{1}{3} [6.25+20.25+12.25+\underset{30.25}{\cancel{42.25}}] = \frac{69}{3} = 23$$

$$S = \begin{bmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cos}(y,x) & \text{cov}(y,y) \end{bmatrix} = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

Step-4: Eigen values & Eigen vectors

$$|S - \lambda I| = 0 \quad -①$$

S - covariance matrix

λ - Eigen values

I - Identity matrix $\quad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$① \Rightarrow \left| \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0 \quad \Rightarrow \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

$$\left| \begin{bmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{bmatrix} \right| = 0$$

$$(14-\lambda)(23-\lambda) - (11\times 11) = 0$$

$$14\times 23 - 14\lambda - 23\lambda + \lambda^2 - 121 = 0$$

$$\lambda^2 - 37\lambda + 322 - 121 = 0$$

ax^2+bx+c

$$\lambda^2 - 37\lambda + 201 = 0$$

$$\frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

$$\frac{37 \pm \sqrt{1369 - 804}}{2} = \frac{37 \pm \sqrt{565}}{2} = \frac{37 \pm 23.76}{2}$$

$$\frac{37+23.76}{2} = \frac{60.76}{2} = 30.38$$

$$\frac{37-23.76}{2} = \frac{13.24}{2} = 6.62$$

$\lambda_1 = 30.38$

$\lambda_2 = 6.62$.

$$(\underbrace{S}_{\text{cov matrix}} - \underbrace{\lambda}_{\text{eigen value } \lambda_1} \overset{\text{identity matrix}}{I})\, U_1 = 0 \text{ — eigenvector}$$

step 5 $\quad \lambda_1 > \lambda_2$

eigen vector for λ_1

$$(S - \lambda_1 I)\, U_1 = 0$$

$$\begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = 0$$

$(14-\lambda_1)U_1 - 11U_2 = 0$ — ①

$-11U_1 + (23-\lambda_1)U_2 = 0$ — ②

① $\Rightarrow (14-\lambda_1)U_1 = 11U_2$

$$\frac{U_1}{U_2} = \frac{11}{14-\lambda_1} = t$$

$U_1 = 11$, $U_2 = 14-\lambda_1$

$$U_1 = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} 11 \\ 14-30.384 \end{bmatrix}$$

Eigen vector for λ_1 $\quad U_1 = \begin{bmatrix} 11 \\ -16.3849 \end{bmatrix}$

Normaliged eigen vector

$$e_1 = \frac{1}{\sqrt{11^2 + (-16.38)^2}} \begin{bmatrix} 11 \\ -16.38 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{11}{\sqrt{11^2 + (-16.38)^2}} \\ \frac{-16.38}{\sqrt{11^2 + (-16.38)^2}} \end{bmatrix} = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$$

$$P_{11} = e_1^T \begin{bmatrix} x_i - \bar{x} \\ y_i - \bar{y} \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} 4-8 \\ 11-8.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} -4 \\ 2.5 \end{bmatrix}$$

$$= 0.5574(-4) - 0.8303(2.5)$$

$$= -2.2296 - 2.07575 = -4.30535$$

$$P_{12} = e_1^T \begin{bmatrix} 8-8 \\ 4-8.5 \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} 0 \\ -4.5 \end{bmatrix}$$

$$= (-0.8303)(4.5) = 3.7363$$

$$P_{13} = e_1^T \begin{bmatrix} 13-8 \\ 5-8.5 \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} 5 \\ -3.5 \end{bmatrix}$$

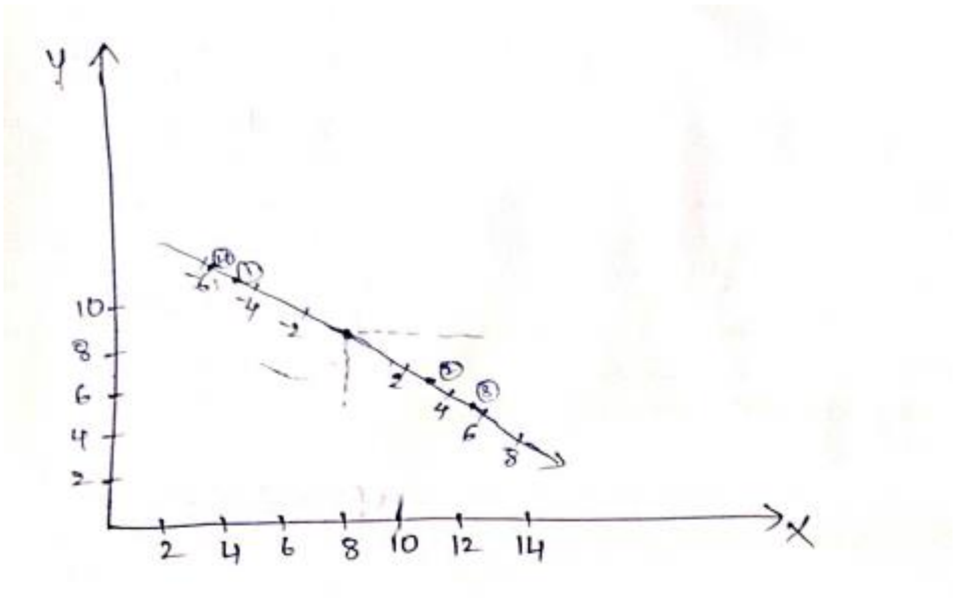
$$= 0.5574 \times 5 + (0.8303)(3.5)$$

$$2.787 + 2.90605 = 5.69305$$

$$P_{14} = e_1^T \begin{bmatrix} 7-8 \\ 14-8.5 \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} -1 \\ 5.5 \end{bmatrix}$$

$$= -0.5574 + 4.56665 = -5.1239$$

PC_1	-4.30535	3.7363	5.69305	-5.1239



3.3 Linear Discriminant Analysis (LDA)

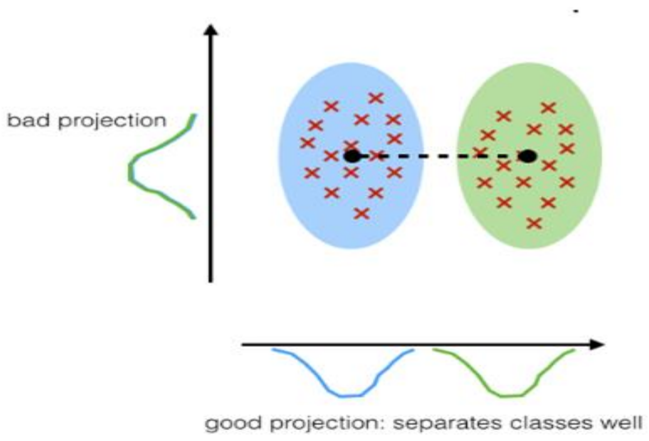
Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

<u>Extension to Linear Discriminant Analysis (LDA)</u>

Linear Discriminant analysis is one of the most simple and effective methods to solve classification problems in machine learning. It has so many extensions and variations as follows:

1. **Quadratic Discriminant Analysis (QDA):** For multiple input variables, each class deploys its own estimate of variance.
2. **Flexible Discriminant Analysis (FDA):** it is used when there are non-linear groups of inputs are used, such as splines.



3. **Flexible Discriminant Analysis (FDA):** This uses regularization in the estimate of the variance (actually covariance) and hence moderates the influence of different variables on LDA.

LDA is a type of Linear combination, a mathematical process using various data items and applying a function to that site to separately analyze multiple classes of objects or items. Fisher's Linear discriminant, linear discriminant analysis can be useful in areas like image recognition and predictive analysis in marketing.

LDA Algorithm

First general steps for performing a Linear Discriminant Analysis

1. Compute the d-dimensional mean vector for the different classes from the dataset.
2. Compute the Scatter matrix (in between class and within the class scatter matrix)
3. Sort the Eigen Vector by decrease Eigen Value and choose k eigenvector with the largest eigenvalue to from a d x k dimensional matrix w (where every column represent an eigenvector)
4. Used **d * k** eigenvector matrix to transform the sample onto the new subspace.
5. This can be summarized by the matrix multiplication.
6. Y = X x W (where X is a **n * d** dimension matrix representing the n samples and **you** are transformed **n * k** dimensional samples in the new subspace.

3.4 Manifold learning

It is a popular and quickly-growing subfield of machine learning based on the assumption that one's observed data lie on a low-dimensional manifold embedded in a higher-dimensional space. This thesis presents a mathematical perspective on manifold learning, delving into the intersection of kernel learning, spectral graph theory, and differential geometry. Emphasis is placed on the remarkable interplay between graphs and manifolds, which forms the foundation for the widely-used technique of manifold regularization.

3.5 Auto encoders and dimensionality reduction in networks

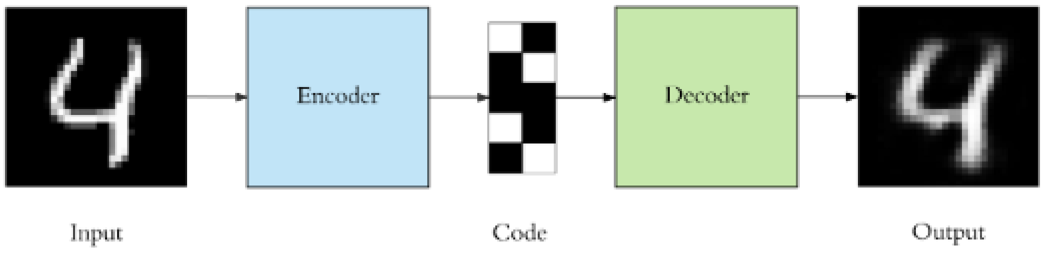
Autoencoder is an unsupervised artificial neural network that compresses the data to lower dimension and then reconstructs the input back. Autoencoder finds the representation of the data in a lower dimension by focusing more on the important features getting rid of noise and redundancy. It's based on Encoder-Decoder architecture, where encoder encodes the high-dimensional data to lower-dimension and decoder takes the lower-dimensional data and tries to reconstruct the original high-dimensional data.

AutoEncoder is an **unsupervised Artificial Neural Network** that attempts to encode the data by compressing it into the lower dimensions (bottleneck layer or code) and then decoding the data to reconstruct the original input. The bottleneck layer (or code) holds the compressed representation of the input data.

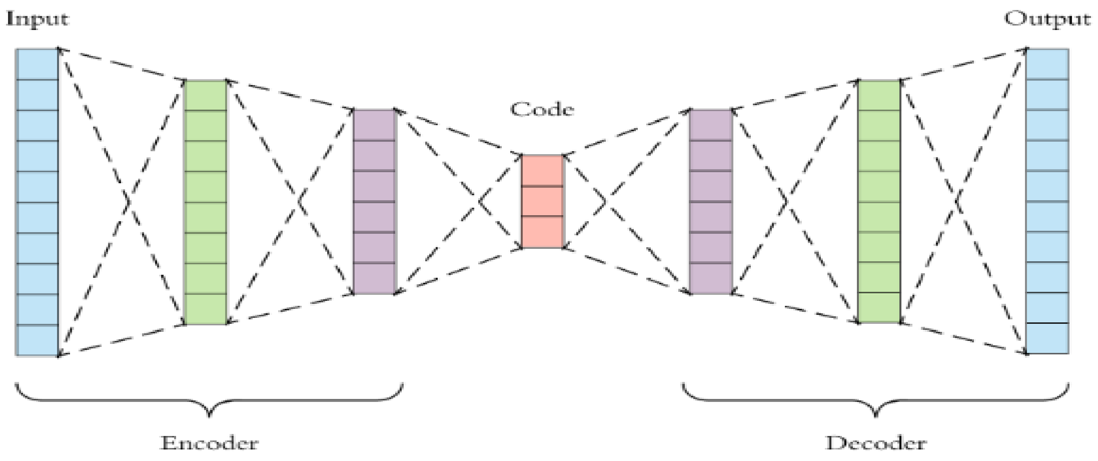
Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation. The code is a compact “summary” or “compression” of the input, also called the *latent-space representation.*

An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.

Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:



- ➢ Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. So we can’t expect an autoencoder trained on



  handwritten digits to compress landscape photos.
- ➢ Lossy: The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation.
- ➢ Unsupervised: To train an autoencoder we don’t need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an *unsupervised* learning technique since they don’t need explicit labels to train on. But to be more precise they are *self-supervised* because they generate their own labels from the training data.

First the input passes through the encoder, which is a fully-connected ANN, to produce the code. The decoder, which has the similar ANN structure, then produces the output only using the code. The goal is to get an output identical with the input. The only requirement is the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

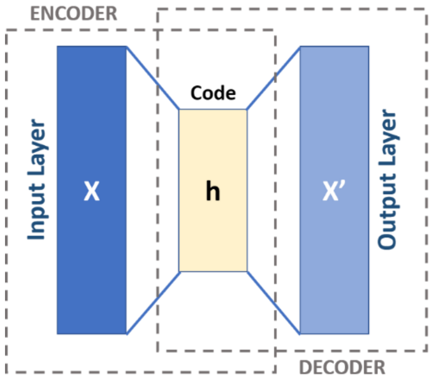
There are 4 hyperparameters that we need to set before training an autoencoder:

- **Code size**: number of nodes in the middle layer. Smaller size results in more compression.
- **Number of layers**: the autoencoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.
- **Number of nodes per layer**: the autoencoder architecture we’re working on is called a *stacked autoencoder* since the layers are stacked one after another.

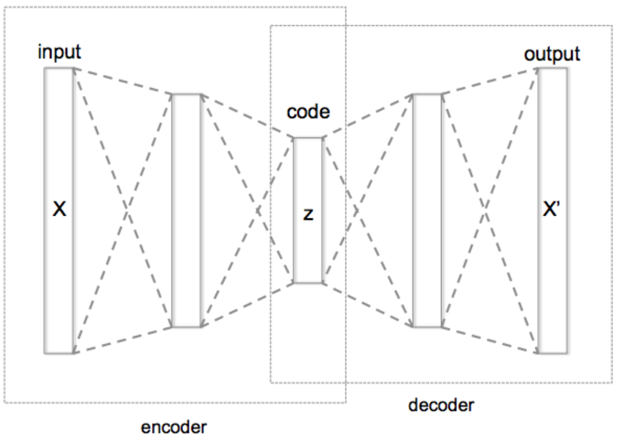
Usually stacked autoencoders look like a "sandwitch". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder.

- **Loss function**: we either use *mean squared error (mse)* or *binary crossentropy*. If the input values are in the range [0, 1] then we typically use crossentropy, otherwise we use the mean squared error.
- Number of layers in encoder and decoder.
- Activation function
- Optimization function

Autoencoders are trained the same way as ANNs via backpropagation.



Schema of a basic Autoencoder



In the above Diagram, X is the input data, z is the lower-dimension representation of input X and X' is the reconstructed input data. The mapping of higher to lower dimensions can be linear or non-linear depending on the choice of the activation function.

Types of AutoEncoders

- Deep Autoencoder
- Sparse Autoencoder
- Under complete Autoencoder
- Variational Autoencoder
- LSTM Autoencoder

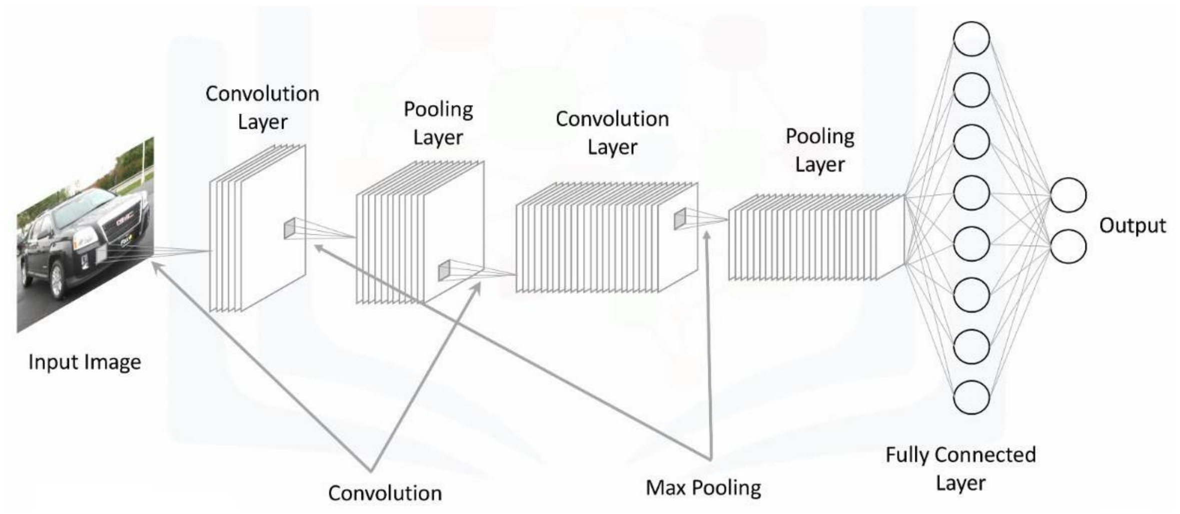
Applications of AutoEncoders

- Dimensionality reduction
- Anomaly detection
- Image denoising
- Image compression
- Image generation

3.6 Introduction to Convnet

A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data.

CNNs are powerful image processing, artificial intelligence (AI) that use deep learning to perform both generative and descriptive tasks, often using machine vison that includes image and video recognition, along with recommender systems and natural language processing (NLP).



3.7 ALEXNETS

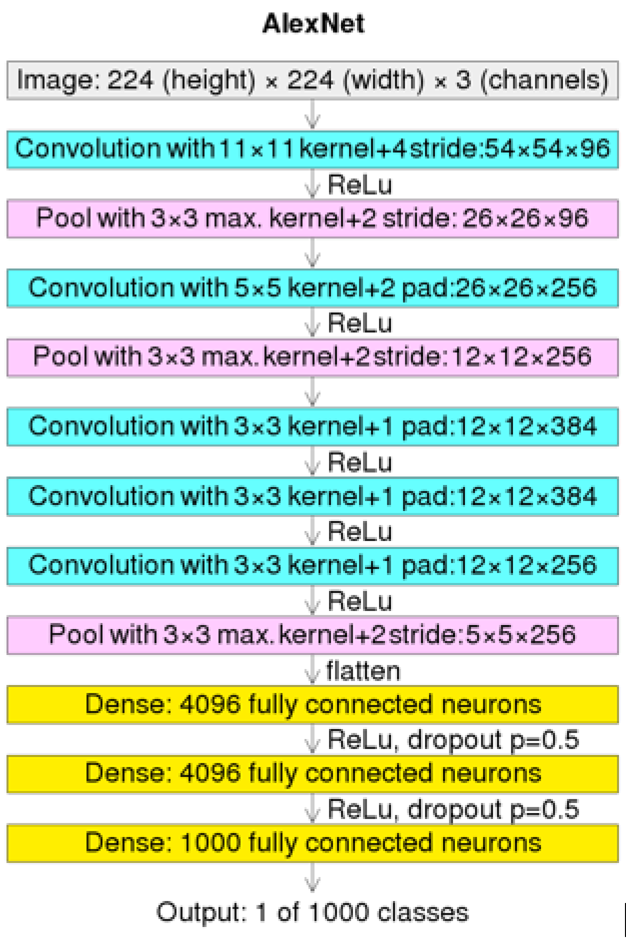
Alexnet won the Imagenet large-scale visual recognition challenge in 2012. The model was proposed in 2012 in the research paper named Imagenet Classification with Deep Convolution Neural Network by Alex Krizhevsky.

The architecture consists of eight layers: five convolutional layers and three fully-connected layers. AlexNet had 60 million parameters, a major issue in terms of overfitting. Two methods were employed to reduce overfitting:

- **Data Augmentation.** The authors used label-preserving transformation to make their data more varied. Specifically, they generated image translations and horizontal reflections, which increased the training set by a factor of 2048. They also performed Principle Component Analysis (PCA) on the RGB pixel values to change the intensities of RGB channels, which reduced the top-1 error rate by more than 1%.

- **Dropout.** This technique consists of "turning off" neurons with a predetermined probability (e.g. 50%). This means that every iteration uses a different sample of the model's parameters, which forces each neuron to have more robust features that can be used with other random neurons. However, dropout also increases the training time needed for the model's convergence.

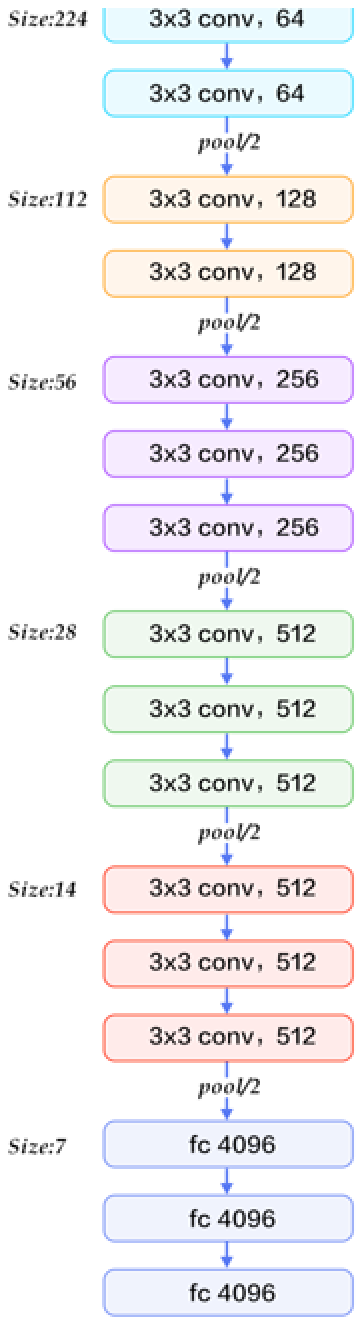
- AlexNet was the first convolution Network which used GPU to boost performance.
- AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer.
- Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU.
- The pooling layers are used to perform max pooling.
- Input size is fixed due to the presence of fully connected layers.
- The input size is mentioned at most of the places as 224x224x3 but due to some padding which happens it works out to be 227x227x3
- AlexNet overall has 60 million parameters.


AlexNet
Image: 224 (height) × 224 (width) × 3 (channels)
Convolution with 11×11 kernel+4 stride:54×54×96
ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
Convolution with 5×5 kernel+2 pad:26×26×256
ReLu
Pool with 3×3 max. kernel+2 stride:12×12×256
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
ReLu
Pool with 3×3 max. kernel+2 stride:5×5×256
flatten
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 1000 fully connected neurons
Output: 1 of 1000 classes

3.8 VGG Visual Geometry Group

The input to VGG based convNet is a 224*224 RGB image. Preprocessing layer takes the RGB image with pixel values in the range of 0–255 and subtracts the mean image values which is calculated over the entire ImageNet training set.There are total of 13 convolutional layers and 3 fully connected layers in VGG16. VGG has smaller filters (3*3) with more depth instead of having large filters. It has ended up having the same effective receptive field as if you only have one 7 x 7 convolutional layers.

Another variation of VGGNet has 19 weight layers consisting of 16 convolutional layers with 3 fully connected layers and same 5 pooling layers. In both variation of VGGNet there consists of two Fully Connected layers with 4096 channels each which is followed by another fully connected layer with 1000 channels to predict 1000 labels. Last fully connected layer uses softmax layer for classification purpose.



Limitations Of VGG 16:

- It is very slow to train (the original VGG model was trained on Nvidia Titan GPU for 2-3 weeks).
- The size of VGG-16 trained imageNet weights is *528* MB. So, it takes quite a lot of disk space and bandwidth which makes it inefficient.
- 138 million parameters lead to exploding gradients problem.

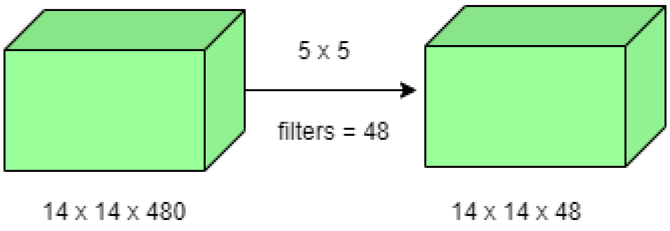
3.9 Inception

Inception Net is a victory over the previous versions of CNN models. It achieves an accuracy of top-5 on ImageNet, it reduces the computational cost to a great extent without compromising the speed and accuracy.

This architecture has *22* layers in total. Using the dimension-reduced inception module, a neural network architecture is constructed. This is popularly known as **GoogLeNet (Inception v1)**. GoogLeNet has *9* such inception modules fitted linearly. It is *22* layers deep (*27*, including the pooling layers). At the end of the architecture, fully connected layers were replaced by a global average pooling which calculates the average of every feature map.

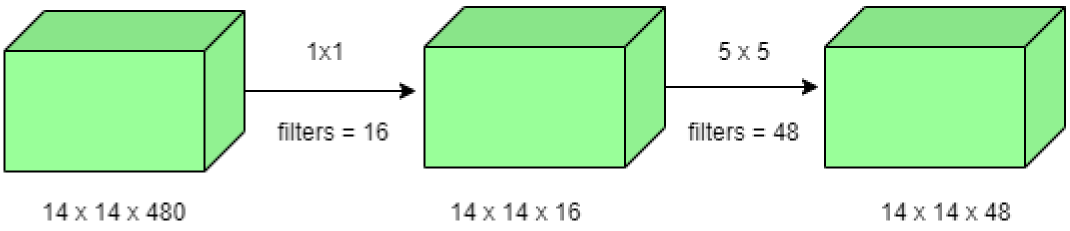
1 X 1 convolution: A 1×1 convolution simply maps an input pixel with all its respective channels to an output pixel. 1×1 convolution is used as a dimensionality reduction module to reduce computation to an extent.

For instance, we need to perform 5×5 convolution without using 1×1 convolution as below:



Number of operations involved here is $(14\times14\times48) \times (5\times5\times480) = 112.9M$

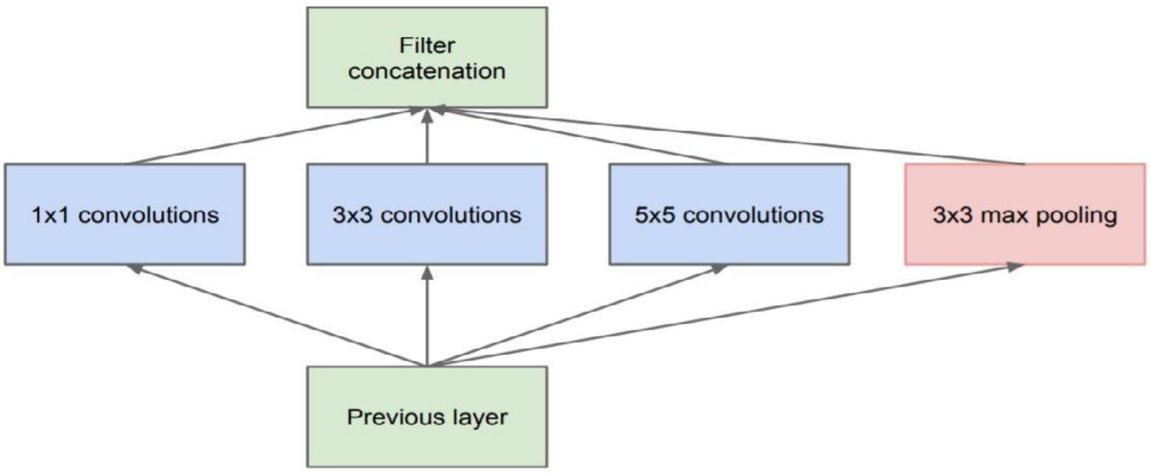
Using 1×1 convolution:



Number of operations for 1×1 convolution $= (14\times14\times16) \times (1\times1\times480) = 1.5M$

Number of operations for 5×5 convolution = *(14×14×48) × (5×5×16) = 3.8M* After addition we get, *1.5M + 3.8M = 5.3M*

Thus, 1×1 convolution can help to reduce model size which can also somehow help to reduce the overfitting problem.



The above depicted Inception module simultaneously performs 1 * 1 convolutions, 3 * 3 convolutions, 5 * 5 convolutions, and 3 * 3 max pooling operations. Thereafter, it sums up the outputs from all the operations in a single place and builds the next feature. The architecture does not follow Sequential model approach where every operation such as pooling or convolution is performed one after the other.

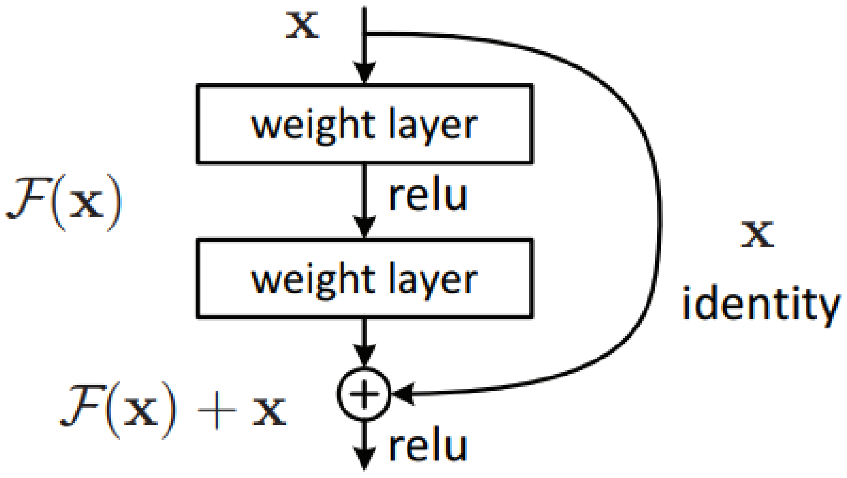
<u>Disadvantages:</u>Larger model models using InceptionNet are prone to overfit especially with limited number of label samples. The model can be biased towards certain classes that have labels present in high volume than the other.

3.10 Residual Network:

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks. In this network, we use a technique called ***skip connections***.

The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. Resnets are made by stacking these residual blocks together.The approach behind this network is instead of layers learning the underlying mapping, we allow the network to fit the residual mapping. So, instead of say H(x), initial mapping, let the network fit,

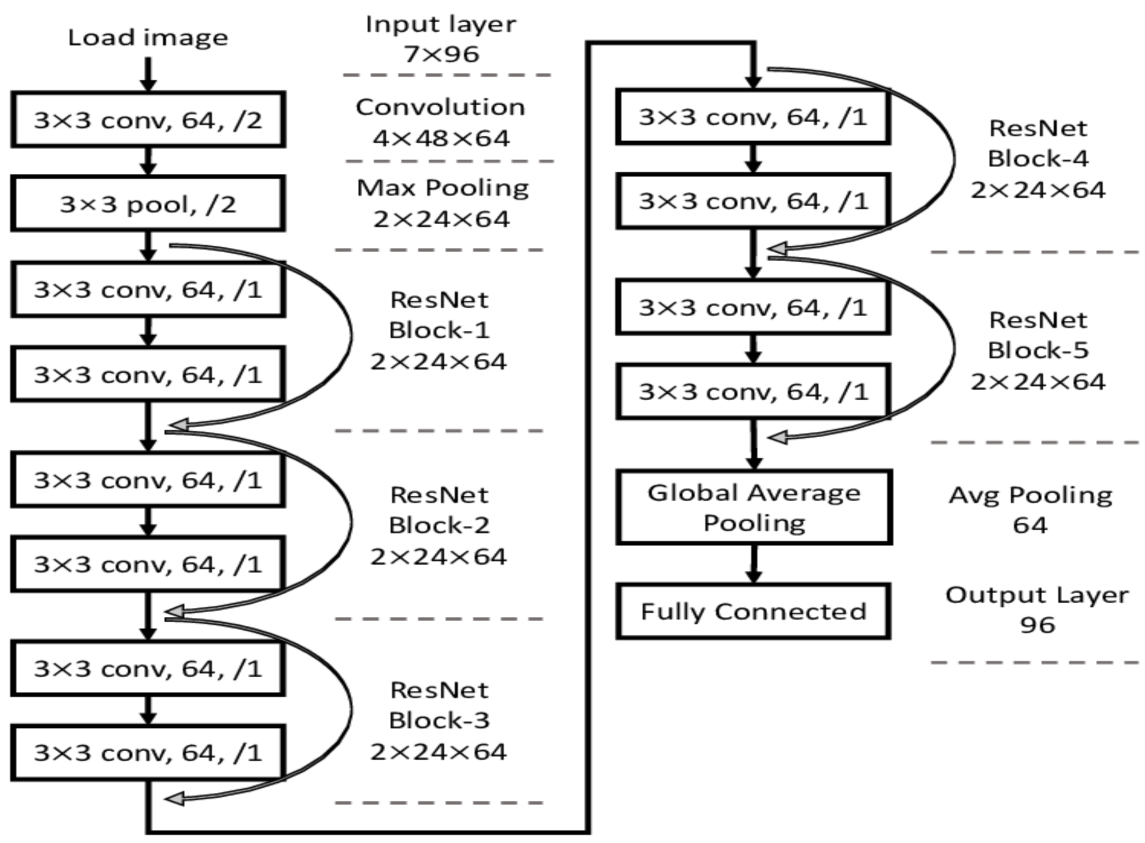
***F(x) := H(x) - x* which gives *H(x) := F(x) + x*.**


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

The advantage of adding this type of skip connection is that if any layer hurt the performance of architecture then it will be skipped by regularization. So, this results in training a very deep neural network without the problems caused by vanishing/exploding gradient.

There is a similar approach called "**highway networks**", these networks also use skip connection. Similar to LSTM these skip connections also use parametric gates.

These gates determine how much information passes through the skip connection. This architecture however has not provided accuracy better than ResNet architecture.



Hence ResNet **uses Batch Normalization at its core**. The Batch Normalization adjusts the input layer to increase the performance of the network. The problem of covariate shift is mitigated.ResNet makes use of the Identity Connection, which helps to protect the network from **vanishing gradient problem**.

3.11HYPER PARAMETER OPTIMIZATION

There are numerous parameters and layers in Deep neural networks. Calculating them and training the network with the parameters are difficult. Hence we need to optimize the hyper parameters.

The hyper parameters to be optimized are :

- Learning rate :
    - Learning rate controls how much to update the weight in the optimization algorithm. To optimize it, use fixed learning rate, decrease learning rate, momentum based methods or adaptive learning rates depending on our choice of optimizer such as SGD.

- Number epochs
  - The number of times the entire training set pass through the neural network is called an epoch. Increase the epoch until the error becomes minimum.

- Batch size
  - ConvNet is sensitive to batch size. 16 to 128 batch size is a good choice.

- Activation function
  - Use activation functions such as ReLU and softmax to optimize.

- Number of hidden layers and units
  - When the neural network has less number of layers , it might to lead to underfitting.

- Weight initialization
  - Initialize weight with small random number to prevent dead neuron. But not too small value to avoid zero gradient.
  - Uniform distribution works well.

- Dropout
  - Dropout probability of 0.5 is well suited for regularization.

3.12 Normalization

Normalization is a pre-processing technique used to standardize data. In other words, having different sources of data inside the same range. Not normalizing the data before training can cause problems in our network, making it drastically harder to train and decrease its learning speed.

There are two main methods to normalize our data. The most straightforward method is to scale it to a range from 0 to 1:

$$X_{normalised}= x\text{-}m/X_{MAX}\text{-}X_{MIN}$$

X the data point to normalize, M the mean of the data set, X_{MAX} the highest value, and X_{MIN} the lowest value. This technique is generally used in the inputs of the data. The non-normalized data points with wide ranges can cause instability in Neural Networks. The relatively large inputs can cascade down to the layers, causing problems such as exploding gradients.

Questions to revise:

Part A:

1. List out the various types of Dimensionality Reduction techniques.
2. The input and output of auto-encoder is same? Justify.
3. Identify the different architectures of ConvNet and its layers split up.
4. List down the hyper parameters to be optimised in neural networks for better performance.
5. Illustrate different metric learning measurements?
6. Define manifold learning.
7. Abbreviate PCA ? List out its advantages.
8. Abbreviate LDA ? List out its advantages.
9. Write down the advantages of Softmax function over other activation functions.
10. How do you normalise a dense convolutional neural network?

Part B:

1. List and explain the steps involved in Dimensionality Reduction using LDA technique.

2. Illustrate and explain in detail the architectures of AlexNet and VGG.

3. Describe the architectures of Inception and ResNet with neat diagram.

4. Sketch the schematic diagram of Autoencoders and explain its workflow.

5. How do you train a ConvNet? Summarize the concept of Weight initialisation and Batch Normalisation.

6. List down the hyper parameters to be optimised in neural networks. Explain how the hyper parameters are optimized.

7. Reduce the number of features of the given data set from two to one.

X	4	8	13	7
Y	11	4	5	14