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

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS

UNIT – III – DEEP LEARNING – SECA4002

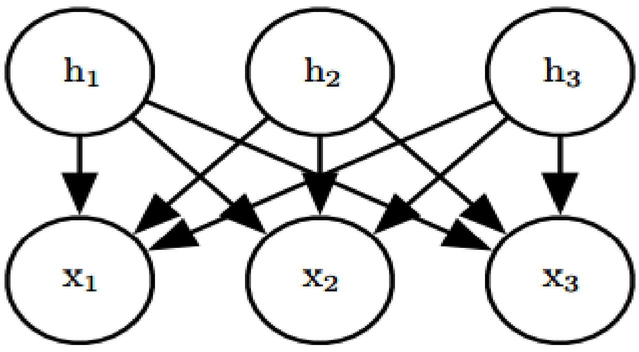
UNIT III DIMENTIONALITY REDUCTION

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 Linear Factor Models:

linear factor models are used as building blocks of mixture models of larger, deep probabilistic models. A linear factor model is defined by the use of a stochastic linear decoder function that generates x by adding noise to a linear transformation of h. It allows us to discover explanatory factors that have a simple joint distribution. A linear factor model describes the data-generation process as follows. (we sample the explanatory factors h from a distribution)

$$h \sim p(h)$$



$$\mathbf{x} = \boldsymbol{W}\mathbf{h} + \boldsymbol{b} + \textbf{noise}$$

Figure:1 Linear Factor Model

3.2 Dimensionality Reduction:

- High dimensionality is challenging and redundant
- It is natural to try to reduce dimensionality
- We reduce the dimensionality by feature combination i.e., combine old features X to create new features Y as given below

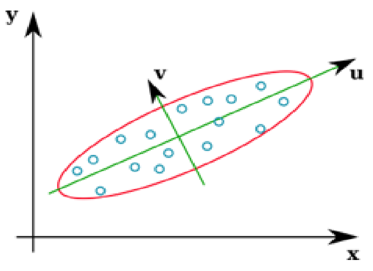
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \rightarrow f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} = y \quad \text{with } k < d$$

Figure 2: Dimensionality Reduction

3.3 Principal Component Analysis (PCA):

Principal Component Analysis, or simply PCA, is a statistical procedure concerned with elucidating the covariance structure of a set of variables. In particular it allows us to identify the principal directions in which the data varies.

For example, in figure 1, suppose that the triangles represent a two variable data set which we have measured in the X-Y coordinate system. The principal direction in which the data varies is shown by the U axis and the second most important direction is the V axis orthogonal to it. If we place the U-V axis system at the mean of the data it gives us a compact representation. If we transform each (X, Y) coordinate into its corresponding (U, V) value, the data is de-correlated, meaning that the co-variance between the U and V variables is zero. For a given set of data, principal component analysis finds the axis system defined by the principal directions of variance (ie the U − V axis system in figure 3). The directions U and V are called the principal components



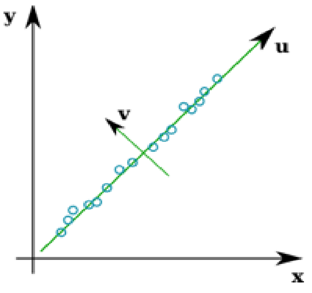


Figure 3A: PCA for Data Representation Figure 3B: PCA Dimension Reduction

If the variation in a data set is caused by some natural property, or is caused by random experimental error, then we may expect it to be normally distributed. In this case we show the nominal extent of the normal distribution by a hyper-ellipse (the two-dimensional ellipse in the example). The hyper ellipse encloses data points that are thought of as belonging to a class. It is drawn at a distance beyond which the probability of a point belonging to the class is low, and can be thought of as a class boundary.

If the variation in the data is caused by some other relationship, then PCA gives us a way of reducing the dimensionality of a data set. Consider two variables that are nearly related linearly as shown in figure 3B. As in figure 3A the principal direction in which the data varies is shown by the U axis, and the secondary direction by the V axis. However in this case all the V coordinates are all very close to zero. We may assume, for example, that they are only non zero because of experimental noise. Thus in the U V axis system we can represent the data set by one variable U and discard V . Thus we have reduced the dimensionality of the problem by 1Computing the Principal Components

3.3.1. Computing the Principal Components

In computational terms the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the co-variance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on. To see how the computation is done we will give a brief review on eigenvectors/eigenvalues.

Let A be a n × n matrix. The eigenvalues of A are defined as the roots of:

$$\textit{Determinant}\,(\boldsymbol{A} - \lambda\boldsymbol{I}) = |(\boldsymbol{A} - \lambda\boldsymbol{I})| = 0$$

where I is the n n identity matrix. This equation is called the characteristic equation (or characteristic polynomial) and has n roots.
Let λ be an eigenvalue of A. Then there exists a vector x such that:

$$\boldsymbol{Ax} = \lambda\boldsymbol{x}$$

The vector x is called an eigenvector of A associated with the eigenvalue λ. Notice that there is no unique solution for x in the above equation. It is a direction vector only and can be scaled to any magnitude. To find a numerical solution for x we need to set one of its elements to an arbitrary value, say 1, which gives us a set of simultaneous equations to solve for the other elements. If there is no solution, we repeat the process with another element. Ordinarily we normalize the final values so that x has length one, that is $x \cdot x^T = 1$.
Suppose we have a 3 × 3 matrix A with eigenvectors x1, x2, x3, and eigenvalues λ1, λ2, λ3 so:

$$Ax1 = \lambda 1x1 \qquad Ax2 = \lambda 2x2 \qquad Ax3 = \lambda 3x3$$

Putting the eigenvectors as the columns of a matrix gives:

$$\boldsymbol{A}\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

writing:

$$\Phi = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \boldsymbol{x}_3 \end{bmatrix} \qquad \Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

gives us the matrix equation:

$$\boldsymbol{A}\Phi = \Phi\Lambda$$

gives us the matrix equation: AΦ = ΦΛ We normalised the eigenvectors to unit magnitude, and they are orthogonal, so: ΦΦT = ΦT Φ = I, which means that: Φ T AΦ = Λ and: A = ΦΛΦT. Now let us consider how this applies to the covariance matrix in the PCA process. Let Σ be an n×n covariance matrix. There is an orthogonal n × n matrix Φ whose columns are eigenvectors of Σ and a diagonal matrix Λ whose diagonal elements are the eigenvalues of Σ, such that $\Phi^T \Sigma\Phi = \Lambda$ We can look on the matrix of eigenvectors Φ as a linear transformation which, in the example of figure 3A transforms data points in the [X, Y] axis system into the [U, V] axis system. In the general case the linear transformation given by Φ transforms the data points into a data set where the variables are uncorrelated. The correlation matrix of the data in the new coordinate system is Λ which has zeros in all the off-diagonal elements.

3.3.2 Steps involved in PCA:

- **Start with data for n observations on p variables**
- **Form a matrix of size n X p**
- **Calculate the Covariance Matrix**

- **Calculate the Eigen vectors and Eigen Values**
- **Choose Principal Component from Feature Vectors**
- **Derive the new Data Set**

3.3.3 PCA Advantages:

1. Removes Correlated Features:

   In a real-world scenario, it is very common that we get thousands of features in our dataset. You cannot run your algorithm on all the features as it will reduce the performance of your algorithm and it will not be easy to visualize that many features in any kind of graph. Hence the data set should be reduced. We need to find out the correlation among the features (correlated variables). Finding correlation manually in thousands of features is nearly impossible, frustrating and time-consuming. PCA performs this task effectively. After implementing the PCA on your dataset, all the Principal Components are independent of one another. There is no correlation among them.

2. Improves Algorithm Performance:

   With so many features, the performance of your algorithm will drastically degrade. PCA is a very common way to speed up your Machine Learning algorithm by getting rid of correlated variables which don't contribute in any decision making. The training time of the algorithms reduces significantly with a smaller number of features. So, if the input dimensions are too high, then using PCA to speed up the algorithm is a reasonable choice.

3. Reduces Overfitting:

   Overfitting mainly occurs when there are too many variables in the dataset. So, PCA helps in overcoming the overfitting issue by reducing the number of features.

4. Improves Visualization:

3.3.4. Disadvantages of PCA

1. **Independent variables become less interpretable:** After implementing PCA on the dataset, your original features will turn into Principal Components. Principal Components are the linear combination of your original features. Principal Components are not as readable and interpretable as original features.

2. **Data standardization is must before PCA:** You must standardize your data before implementing PCA, otherwise PCA will not be able to find the optimal Principal Components.

3. **Information Loss:** Although Principal Components try to cover maximum variance among the features in a dataset, if we don't select the number of Principal Components with care, it may miss some information as compared to the original list of features.

3.4 Linear Discrimination Analysis (LDA):

Linear Discriminant Analysis as its name suggests is a linear model for classification and dimensionality reduction. Most commonly used for feature extraction in pattern classification problems.

3.4.1 Need for LDA:

- Logistic Regression is perform well for binary classification but fails in the case of multiple classification problems with well-separated classes. While LDA handles these quite efficiently.
- LDA can also be used in data pre-processing to reduce the number of features just as PCA which reduces the computing cost significantly**.**

3.4.2. Limitations:

- Linear decision boundaries may not effectively separate non-linearly separable classes. More flexible boundaries are desired.
- In cases where the number of observations exceeds the number of features, LDA might not perform as desired. This is called Small Sample Size (SSS) problem. Regularization is required.

Linear Discriminant Analysis or **Normal Discriminant Analysis** or **Discriminant Function Analysis** is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modeling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping. So, we will keep on increasing the number of features for proper classification.

3.4.3 Steps involved in LDA:

There are the three key steps.

(i) Calculate the separability between different classes. This is also known as between-class variance and is defined as the distance between the mean of different classes.

(ii) Calculate the within-class variance. This is the distance between the mean and the sample of every class.

(iii) Construct the lower-dimensional space that maximizes Step1 (between-class variance) and minimizes Step 2(within-class variance).

3.4.4. Pros & Cons of LDA

Advantages of LDA:

1. Simple prototype classifier: Distance to the class mean is used, it's simple to interpret.
2. Decision boundary is linear: It's simple to implement and the classification is robust.
3. Dimension reduction: It provides informative low-dimensional view on the data, which is both useful for visualization and feature engineering.

Shortcomings of LDA:

1. Linear decision boundaries may not adequately separate the classes. Support for more general boundaries is desired.

2. In a high-dimensional setting, LDA uses too many parameters. A regularized version of LDA is desired.

3. Support for more complex prototype classification is desired.

3.5. Manifold Learnings:

- ➢ Manifold learning for dimensionality reduction has recently gained much attention to assist image processing tasks such as **segmentation, registration, tracking, recognition, and computational anatomy.**
- ➢ The drawbacks of PCA in handling dimensionality reduction problems for **non-linear weird** and **curved shaped surfaces** necessitated development of more advanced algorithms like **Manifold Learning**.
- ➢ There are different variant's of Manifold Learning that solves the problem of **reducing data dimensions** and feature-sets obtained from real world problems representing uneven weird surfaces by sub-optimal data representation.
- ➢ This kind of data representation **selectively chooses data points** from a low-dimensional manifold that is embedded in a high-dimensional space in an attempt to **generalize linear frameworks** like PCA.
- ❖ Manifolds give a look of flat and featureless space that behaves like Euclidean space. Manifold learning problems are **unsupervised** where it learns the **high-dimensional structure** of the data from the data itself, without the use of **predetermined classifications** and **loss of importance of information** regarding some characteristic of the original variables.
- ❖ The goal of the manifold-learning algorithms is to recover the o**riginal domain structure**, up to some **scaling** and rotation. The nonlinearity of these algorithms allows them to reveal the domain structure even when the manifold is not **linearly embedded**. It uses some **scaling** and **rotation** for this purpose.
- ❖ *Manifold learning algorithms are divided in to two categories:*
  - ➢ **Global methods:** Allows high-dimensional data to be mapped from high-dimensional to low-dimensional such that the global properties are preserved. Examples include **Multidimensional Scaling (MDS), Isomaps** covered in the following sections.
  - ➢ **Local methods:** Allows high-dimensional data to be mapped to low dimensional such that local properties are preserved. Examples are **Locally linear embedding (LLE), Laplacian eigenmap (LE), Local tangent space alignment (LSTA), Hessian Eigenmapping (HLLE)**
- ➢ **Three popular manifold learning algorithms:**
  - ❑ IsoMap (Isometric Mapping)

Isomap seeks a lower-dimensional representation that maintains 'geodesic distances' between the points. A geodesic distance is a generalization of distance for curved surfaces. Hence, instead of measuring distance in pure Euclidean distance with the Pythagorean theorem-derived distance formula, Isomap optimizes distances along a discovered manifold

- Locally Linear Embeddings

Locally Linear Embeddings use a variety of tangent linear patches (as demonstrated with the diagram above) to model a manifold. It can be thought of as performing a PCA on each of these neighborhoods locally, producing a linear hyperplane, then comparing the results globally to find the best nonlinear embedding. The goal of LLE is to 'unroll' or 'unpack' in distorted fashion the structure of the data, so often LLE will tend to have a high density in the center with extending rays

- t-SNE

t-SNE is one of the most popular choices for high-dimensional visualization, and stands for **t-distributed Stochastic Neighbor Embeddings**. The algorithm converts relationships in original space into t-distributions, or normal distributions with small sample sizes and relatively unknown standard deviations. This makes t-SNE very sensitive to the local structure, a common theme in manifold learning. It is considered to be the go-to visualization method because of many advantages it possesses.

3.6.Auto Encoders:

AutoEncoder is an **unsupervised Artificial Neural Network** that attempts to encode the data by compressing it into the lower dimensions (bottlenecklayer or code) and then decoding the data to reconstruct the original input.The bottleneck layer (or code) holds the compressed representation of theinput data. In AutoEncoder the number of output units must be equal to the number ofinput units since we're attempting to reconstruct the input data.

AutoEncoders usually consist of an encoder and a decoder. The encoder encodes the provided data into a lower dimension which is the size of thebottleneck layer and the decoder decodes the compressed data into itsoriginal form.The number of neurons in the layers of the encoder will be decreasing as we move on with further layers, whereas the number of neurons in the layers of the decoder will be increasing as we move on with further layers. There are three layers used in the encoder and decoder in the following example. The encoder contains 32, 16, and 7 units in each layer respectively and the decoder contains 7, 16, and 32 units in each layer respectively. The code size/the number of neurons in bottle-neck must be less than the number of features in the data. Before feeding the data into the AutoEncoder the data must definitely be scaled between 0 and 1 using MinMaxScaler since we are going to use sigmoid activation function in the output layer which outputs values between0 and 1.When we are using AutoEncoders for dimensionality reduction we'll beextracting the bottleneck layer and use it to reduce the dimensions. Thisprocess can be viewed as **feature extraction**.

The type of AutoEncoder that we're using is **Deep AutoEncoder**, where theencoder and the decoder are symmetrical. The Autoencoders don't necessarily have a symmetrical encoder and decoder but we can have the encoder and decoder non-symmetrical as well.

3.6.1. Types of AutoEncoders are,

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

3.6.2. Hyperparameters of an AutoEncoder

✶ Code size or the number of units in the bottleneck layer
✶ Input and output size, which is the number of features in the data
✶ Number of neurons or nodes per layer
✶ Number of layers in encoder and decoder.
✶ Activation function
✶ Optimization function

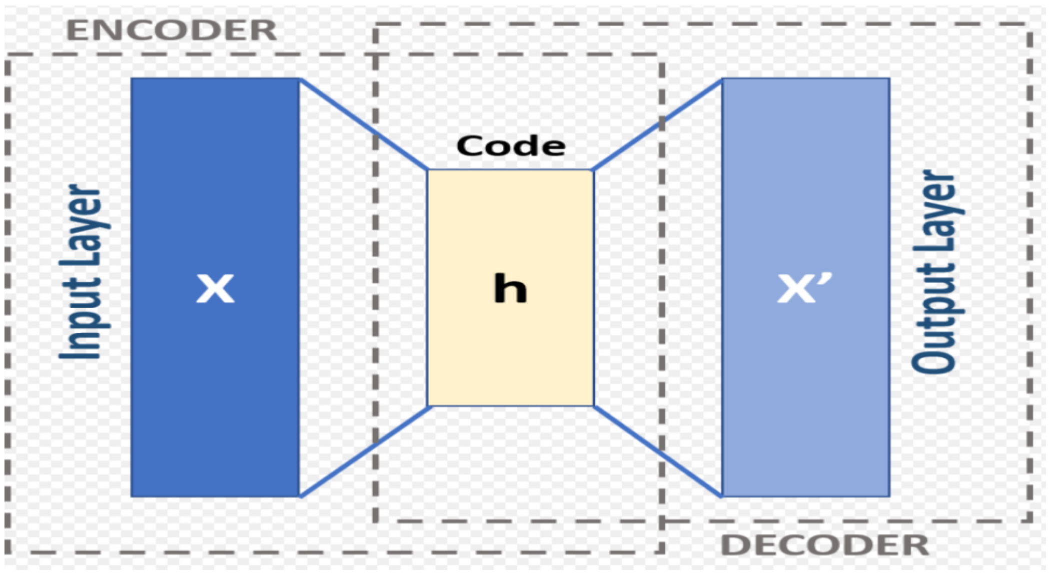


Figure 4: Auto Encoders

3.7. AlexNet:

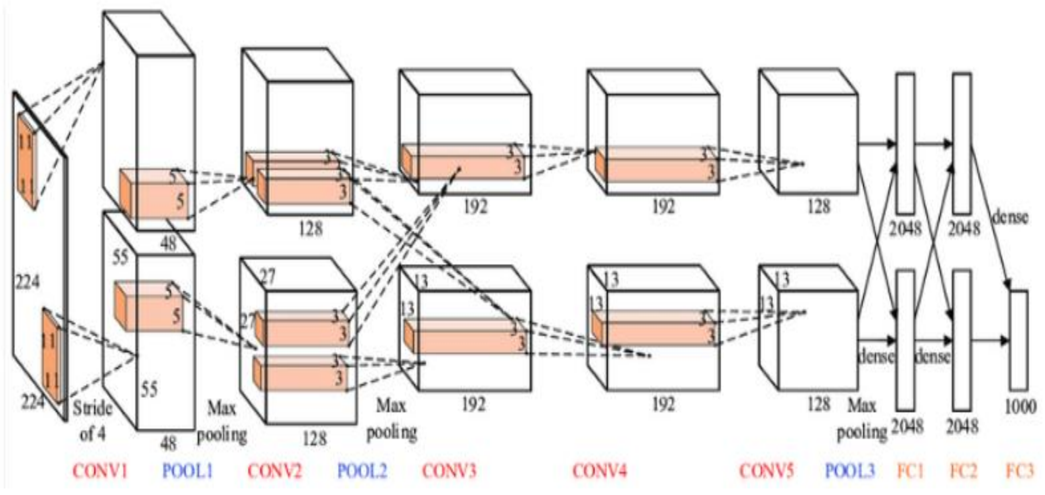


Figure 5: Alexnet Architecture

Alexnet model was proposed in 2012 in the research paper named Imagenet Classification with Deep Convolution Neural Network by Alex Krizhevsky and his colleagues

- The Alexnet has eight layers with learnable parameters
- The model has five layers with a combination of max pooling followed by 3 fully connected layers
- The fully connected layers use Relu activation except the output layer
- They found out that using the Relu as an activation function accelerated the speed of the training process by almost six times.
- They also used the dropout layers, which prevented the model from overfitting.
- The model is trained on the Imagenet dataset. The Imagenet dataset has arounf 14 million images across a 1000 classes.
- The input to this model is the images of size 227X227X3
- The first convolution layer with 96 filters of size 11X11 with stride 4
- The activation function used in this layer is relu. The output feature map is 55X55X96
- Next, we have the first Maxpooling layer, of size 3X3 and stride 2
- Next the filter size is reduced to 5X5 and 256 such filtersare added
  The stride value is 1 and padding 2. The activation function used is again relu. The output size we get is 27X27X256
- Next we have a max-pooling layer of size 3X3 with stride 2. The resulting feature map size is 13X13X256
- The third convolution operation with 384 filters of size 3X3 stride 1 and also padding 1is done next. In this stage the activation function used is relu. The output feature map is of shape 13X13X384

- Then the fourth convolution operation with 384 filters of size 3X3. The stride value along with the padding is 1.The output size remains unchanged as 13X13X384.
- After this, we have the final convolution layer of size 3X3 with 256 such filters. The stride and padding are set to 1,also the activation function is relu. The resulting feature map is of shape 13X13X256

  If we look at the architecture now, the number of filters is increasing as we are going deeper. Hence more features are extracted as we move deeper into the architecture. Also, the filter size is reducing, which means a decrease in the feature map shape.

3.8. VGG-16

- The major shortcoming of too many hyper-parameters of AlexNet was solved by VGG Net by replacing large kernel-sized filters (11 and 5 in the first and second convolution layer, respectively) with multiple 3×3 kernel-sized filters one after another.
- The architecture developed by Simonyan and Zisserman was the 1st runner up of the Visual Recognition Challenge of 2014.
- The architecture consist of 3*3 Convolutional filters, 2*2 Max Pooling layer with a stride of 1.
- Padding is kept same to preserve the dimension.
- There are 16 layers in the network where the input image is RGB format with dimension of 224*224*3, followed by 5 pairs of Convolution(filters: 64, 128, 256,512,512) and Max Pooling.
- The output of these layers is fed into three fully connected layers and a softmax function in the output layer.
- In total there are 138 Million parameters in VGG Net

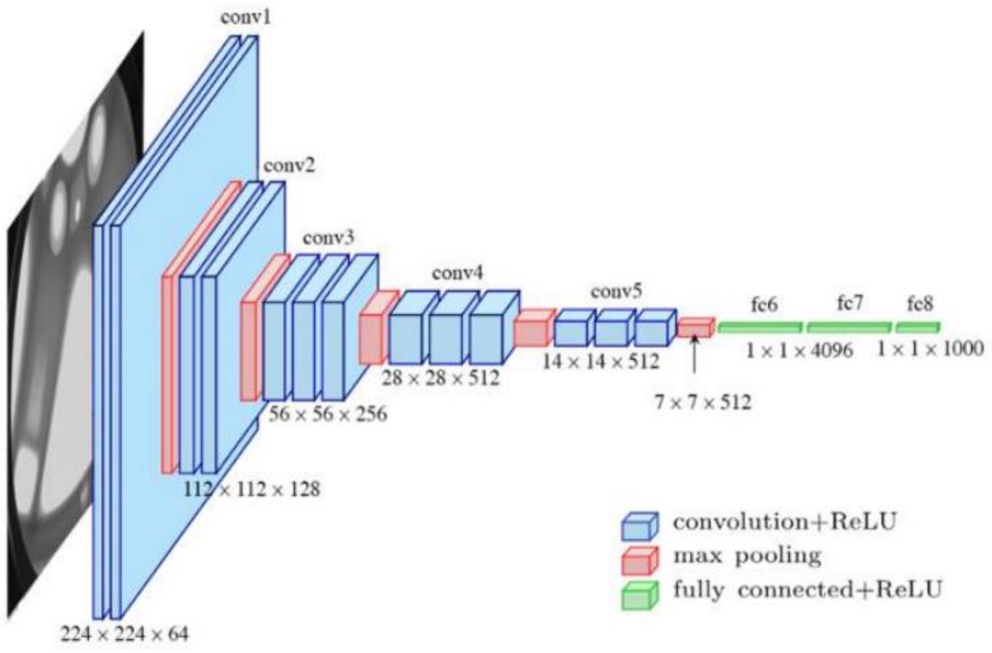


Figure6: VGG Architecture

3.9 ResNet:

ResNet, the winner of ILSVRC-2015 competition is a deep network with over 100 layers. Residual networks (ResNet) is similar to VGG nets however with a sequential approach they also use “Skip connections” and “batch normalization” that helps to train deep layers without hampering the performance. After VGG Nets, as CNNs were going deep, it was becoming hard to train them because of vanishing gradients problem that makes the derivate infinitely small. Therefore, the overall performance saturates or even degrades. The idea of skips connection came from highway network where gated shortcut connections were used

3.10 Inception Net:

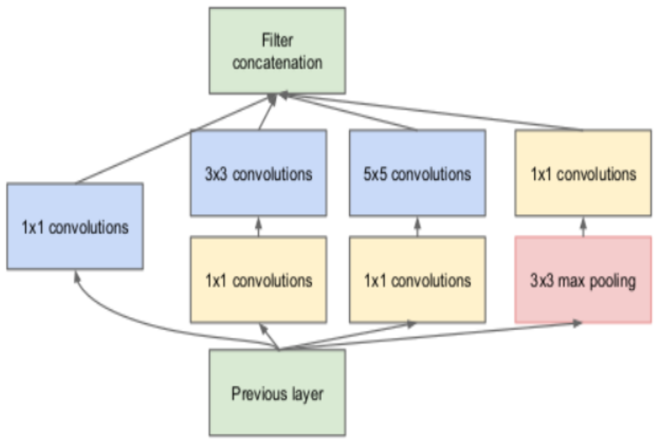


Figure 7: InceptionNet

Inception network also known as GoogleLe Net was proposed by developers at google in “Going Deeper with Convolutions” in 2014. The motivation of InceptionNet comes from the presence of sparse features Salient parts in the image that can have a large variation in size. Due to this, the selection of right kernel size becomes extremely difficult as big kernels are selected for global features and small kernels when the features are locally located. The InceptionNets resolves this by stacking multiple kernels at the same level. Typically it uses 5*5, 3*3 and 1*1 filters in one go.

3.11. Hyperparameter Optimization:

Hyperparameter optimization in machine learning intends to find the hyperparameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyperparameters, in contrast to model parameters, are set by the machine learning engineer before training. The number of trees in a random forest is a hyperparameter while the weights in a neural network are model parameters learned during training. Hyperparameter optimization finds a combination of hyperparameters that returns an optimal model which reduces a predefined loss function and in turn increases the accuracy on given independent data

3.11.1 Hyperparameter Optimization methods

- ❑ Manual Hyperparameter Tuning
- ❑ Grid Search
- ❑ Random Search
- ❑ Bayesian Optimization
- ❑ Gradient-based Optimization

Reference Books:

1. B. Yegnanarayana, "Artificial Neural Networks" Prentice Hall Publications.
2. Simon Haykin, "Artificial Neural Networks", Second Edition, Pearson Education.
3. Laurene Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications", Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

**************************** ALL THE BEST **