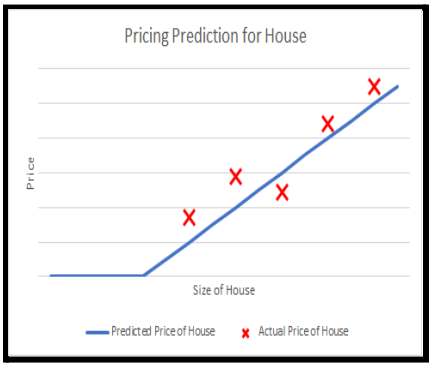
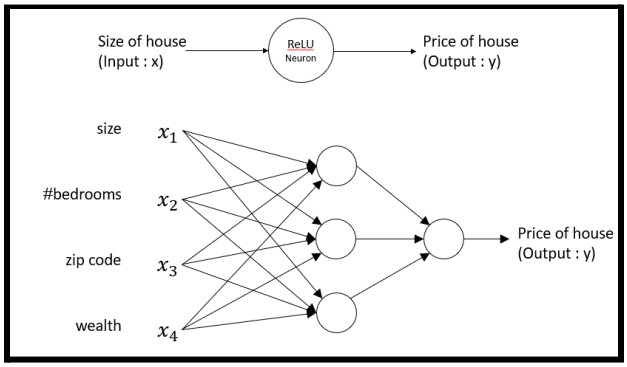
Deep Learning Tech Report

By: Hardik Tibrewal(18EC10020), Tanay Raghavendra(18EC10063), Ayan Chakraborty(18EC10075), Debjoy Saha(18EC30010)

What is a neural network?

The term, Deep Learning, refers to training Neural Networks, sometimes very large Neural Networks. So what exactly is a Neural Network? To get an intuition of this we can consider a Housing Price Prediction example. We have a data set with six houses, and we know the size of the houses in square feet and we know the price of the house and we want to fit a function to predict the price of the houses, the function of the size. We can easily fit a straight line to the points of the data set by using an error metric and minimising it. We then cut this straight line off at the point where the price becomes negative, as the price can never be negative. Hence, finally, we have a linear predictor of housing prices depending on their sizes. However, because we fit the zero at the point where the price becomes negative, the model is not linear. Hence we can think of this function that we've just fit the housing prices as a very simple neural network. As depicted in the figure below, we have as the input to the neural network the size of a house which one we call 'x'. It goes into this node, and then it outputs the price which we call 'y'. The node is a single neuron in a neural network that implements the predictor function. Mathematically, what this neuron does is it inputs the size, computes this linear function, takes a max of zero, and then outputs the estimated price. This is similar to a ReLU function which stands for rectified linear units. A larger neural network is formed by taking many similar single neurons and stacking them together. Let's see an example of multiple neurons neural networks. Now we have multiple features to predict the housing price. Let these new features be size, number of bedrooms, zip code of the area, the wealth of area. Now as depicted in figure 3 we can form multiple ReLU or some other functions' nodes. These ReLU nodes can finally combine using some weights to a final node that would then predict y. As you can see, by stacking together a few of the single neurons or the simple predictors we now have a slightly larger neural network. It is easy to manage a neural network because we only need to give the input x and it output y for the training set. Everything else the network figures out itself, giving us the best model formed from the defined layers in the process. Each of the nodes discussed previously are called hidden units in the neural network and each of them takes its inputs from all four input features. The remarkable thing about neural networks is that, given enough data about x and y, given enough training examples with both x and y, neural networks are remarkably good at figuring out functions that accurately map from x to y.

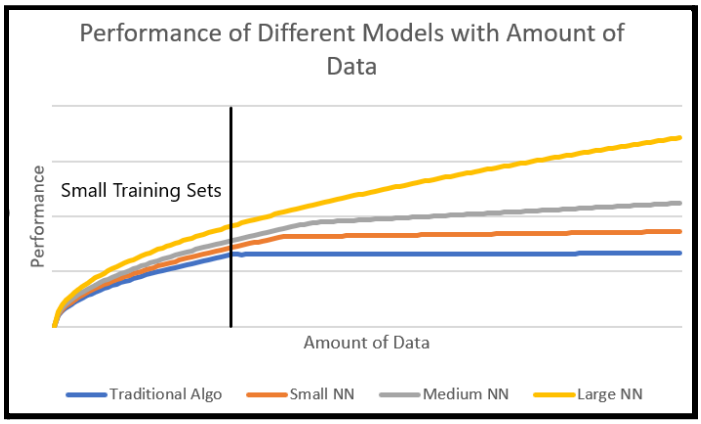




Why is Deep Learning taking off now?

The modules for deep learning and neural networks have existed for a long time. The reason that deep learning has become so popular in the recent past is that as society has digitized, more and more data has come in to solve the same problems that existed before. As we can see in the approximate graph below, deep learning models do not plateau as more data is fed in the model, or more accurately they plateau slower. This means that we can reach higher levels of accuracy with more data in a deep learning model. Hence because of the digitization of a society where so much of our activity is now in the digital realm, we spend so much time on the computers on

websites on mobile apps and activities on digital devices creates data and thanks to the rise of inexpensive cameras built into our cell phones accelerometers all sorts of sensors in the Internet of Things we have been collecting more and more data The data is too large for traditional learning algorithms to be able to effectively. However, if we want to hit this very high level of performance then we need two things, first, we need the ability to train a big enough neural network to take advantage of the huge amount of data and second is to make sure we are collecting huge amounts of useful data.



Modern-Day Applications of Deep Learning

Some applications of Deep Learning include :

1. Self Driving Cars: DL is the driver behind autonomous driving.
2. Natural Language Processing: NLP applications primarily use DL and have wide applications, most prominently in Fake News/Hate Speech Detection.
3. Visual Recognition: DL is used to form visual recognition which has helped law enforcement bodies catch criminals quicker. It has several other applications.
4. Vast Improvement of already existing supervised models: Apart from these things mentioned above, DL has helped us improve the performance of day-to-day prediction models to help in real-life problems.

Some Basic Neural Network Concepts

Here are some standard notations used in deep learning.

1. Binary Classification: A classification with only two classes. Usually represented using the two binary digits. (1 - is a cat, 0 is not a cat)
2. Binary Step Function : $f(x) = \{1\ if\ x >= 0,\ 0\ if\ x < 0\}$
3. Linear Function : $f(x) = ax$
4. Sigmoid Function : $\sigma(z) = \frac{1}{1+e^{-z}}$
5. Tanh Function : $tanh(x) = \frac{1-e^{2x}}{1+e^{2z}}$

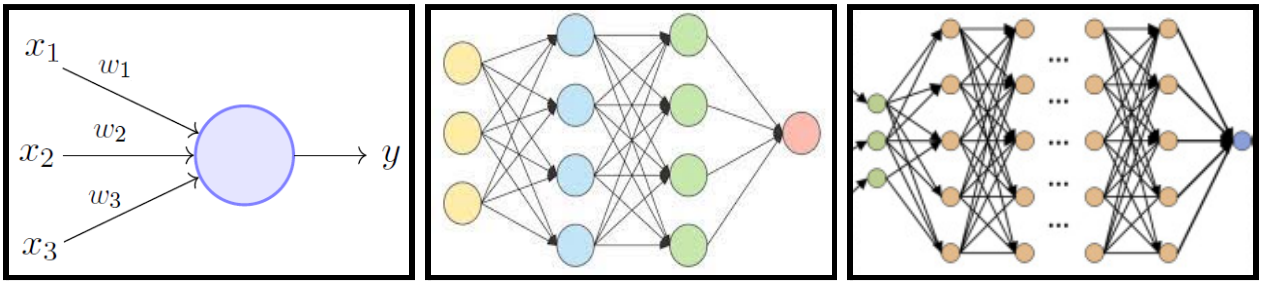
6. ReLU : $f(x) = max(0, x)$
7. Leaky ReLU : $f(x) = \{x\ if\ x >= 0,\ 0.1x\ if\ x < 0\}$
8. Parameterized ReLU : $f(x) = \{x\ if\ x >= 0,\ ax\ if\ x < 0\}$
9. Exponential Linear Unit : $f(x) = \{x\ if\ x >= 0,\ a(e^{x} - 1)\ if\ x < 0\}$
10. Swish Function : $f(x) = \frac{x}{1+e^{-x}}$
11. Logistic Regression: Logistic regression is a learning algorithm used in a supervised learning problem when the output *y* are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data.
12. Logistic Regression Model: $\hat{y} = \sigma(w^{T}x + b)$ where σis the sigmoid function.
13. Logistic Regression Cost: $J(w, b) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)})]$ (we iterate on w,b to optimise this.)
14. Gradient Descent : $w := w - \alpha\frac{d(J(w,b))}{d(w)}$; $b := b - \alpha\frac{d(J(w,b)}{d(b)}$ note: the derivatives are partial derivatives

Deep Neural Networks - The Model of Deep Learning

Motivation for choosing deep neural networks over shallow networks:

For deep learning, artificial neural networks are used to learn the desired function. An artificial neural network consists of multiple layers of the individual computational structure called a perceptron. Multiple perceptrons together create a neural network layer, which can be used to create a neural network. The structure of the resultant graph, made by stacking perceptrons into layers and joining them, resembled the neurons of the brain, giving rise to the name neural network. Traditional neural networks only have a few layers, but deep neural networks can have 100 or even more layers.

Single Perceptron → Basic Neural Network → Deep Neural Network



Previous work in machine learning showed that a single linear perceptron cannot be a universal classifier, but a network with a non-polynomial activation function with a single layer of unbounded width can be a universal classifier. However, the unbounded width poses a problem since unbounded widths may result in matrices that may

be too large to handle during computation. Deep learning is a modern variation that serves as a practical approach to this issue, where the neural network can have a large number of layers (or a “deep” structure), and each layer can have a bounded width. This allows for practical application to the issue and learning of more complex functions, while still serving as a universal classifier in mild conditions.

In the time soon after the discovery of neural networks, deep neural networks were not feasible to implement since they required a large amount of data and processing power. The reason for requiring excessive computing power is obvious, the large number of matrix and transcendental operations that need to be carried out for each iteration of the process. The reason why a lot of data is required is related to overfitting. The double-edged sword offered by deep neural networks is that they are quite capable of identifying highly complex functions but this makes them prone to overfitting the training data. One of the approaches of reducing overfitting is to simply provide more data, thus forcing the learned function to generalise better, since a greater number of points from the distribution will need to be fit. Modern GPUs allow for parallel computation and provide a large number of computation units which makes the process of training faster, whereas the mass adoption of the Internet and advanced techniques allow for large scale data-mining.

Benefits of Deep Neural Networks and their Explainability:

As mentioned previously, deep neural networks were built upon the shallow artificial neural networks used in traditional machine learning, which were themselves built upon the concept of a perceptron, which forms a single “neuron” of our neural network.

A single perceptron takes an input, uses its characteristic weights to perform a vector-vector dot product (or matrix-vector product in case of inputs in a batch) and then uses an activation function for classification. Since the input space is restricted to a set number of dimensions, this process is akin to choosing a hyperplane in the input space, capable of performing binary classification on the data.

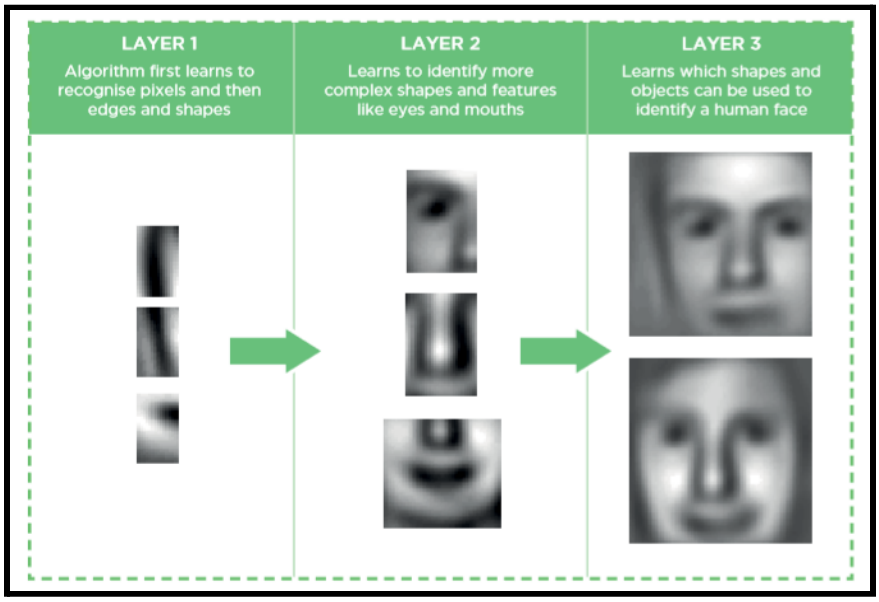
A single layer of neurons is choosing a set of hyperplanes in the input space, which can be algebraically represented as a function of the space. Stacking multiple layers allows us to learn functions of functions, allowing the network to extract more information from the data.

Thus, a deep neural network is capable of learning much more complex functions, since each additional layer learns a function of a function. Each layer of the network extracts some more information from the data, which generally helps in better learning. Some of the intermediate results may be understandable by a human, but this is neither a requirement (in most cases) nor a guarantee.

This feature of deep neural networks is especially useful for extracting information from complex data, such as images, video, audio, etc. They contain a vast amount of information that can be easily identified by a human but cannot be represented mathematically easily, making processing more difficult. In such cases, deep neural networks can be used to learn extremely complex functions which can extract information from the input step-by-step.

A popular example of this step-by-step feature extraction is in the field of image processing and object identification. Images, for humans, are easy to process and identify. However, images are stored as a tensor, where each pixel is described by its red-green-blue colour intensity, which ranges from 0 to 255. This makes it difficult to mathematically pin down what constitutes a particular object, say a human face when an image is represented in such a form. With deep neural networks, each layer can focus on extracting a certain kind of information from the image. For example, the first layer can extract edges from the image, then use the edges to identify basic shapes, and eventually build upon it to individually identify humans from their photographs. The astonishing part of deep learning is that the deep neural network extracts these basic features on its own, without being explicitly coded to do

so. The normal approach of just using the loss function on the final layer is taken here, but still, the network can learn the underlying patterns completely on its own, which is what makes deep learning so useful.
The theory behind the extraction of such features is explained later while explaining Convolutional Neural Networks
Below is a figure illustrating how each layer builds on top of the first layer.



Convolutional Neural Networks - Deep Learning for Images

Infeasibility of directly using Deep Neural Networks on Images:

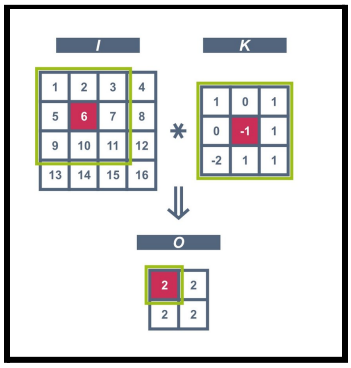
Deep neural networks cannot be directly applied to images. To illustrate the infeasibility of using Deep Neural Networks directly on images, we consider the following example. Let us consider 64 by 64 images. So that's 64 by 64 by 3 because there are three colour channels. And if we multiply that out, that's 12288. So ‘X’ the input feature has dimension 12288. And that's not too bad. But 64 by 64 is a very small image. If we work with larger images, such as a 1000 pixel by 1000 pixel image, that's just one megapixel. But the dimension of the input features will be 1000 by 1000 by 3, because we have three RGB channels, and that's three million. But if we have three million input features, then this means that ‘X’ here will be three million dimensional. And so, if in the first hidden layer we have just 1000 hidden units, then the total number of weights that is the matrix W1, this matrix will be 1000 by 3 million dimensional. And this means that this matrix will have three billion parameters which is just very large. And with that many parameters, it's difficult to get enough data to prevent a neural network from overfitting. And also, the computational requirements and the memory requirements to train a neural network with three billion parameters is infeasible.

Convolutional Neural Networks are a modification to Deep Neural networks taking their inspiration from the Signal Processing domain. Instead of directly applying Multiply and Add operations, each layer implements the Convolution Operation on its input feature vector. Its theory and advantages are explained in more detail below

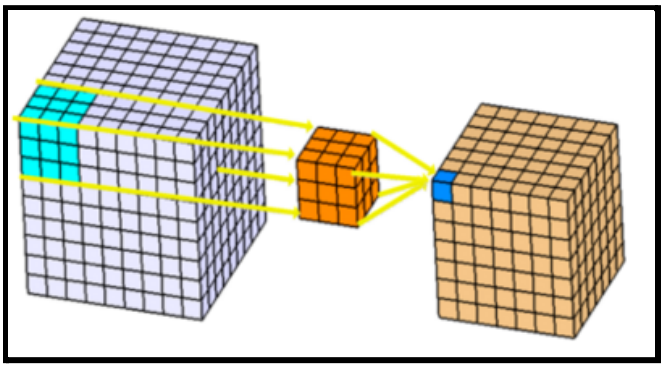
The theory behind Convolution:

1D convolution of two signals is generated by sliding 1 signal over the entire length of the other signal, and at each time step, calculating the element-wise product and then summing it up to generate the corresponding value at that time step. So, we are weighting the 1st signal values with the 2nd signal values. 2D convolution is done with

two 2D matrices. In the area of image processing, 1 input matrix is generally the input image (consider M by N, and single-channel) and the 2nd matrix is the weight matrix (K1 by K2), which usually has much smaller dimensions. The weight matrix is placed over a subpart of the image (K1 by K2), and element-wise multiplication between the weight matrix and the input image is carried out over that subpart and summed up to generate an equivalent pixel for that subpart. This is the same as the process of applying filter kernels to images in traditional Image Processing. The filter kernels are nothing but the weight matrices. 3D convolution is just the extension of 2D convolution in 3 dimensions, in which both the filter kernel and the Image is 3D (with their Z dimensions matching). The subpart to be considered in this case will be a 3D block with the same shape as the 3D filter kernel. Element wise multiplication is again carried out between the kernel and the subpart. In this case, we will get (N*K1*K2) elements, where N is equal to the number of channels (the Z dimension of the image), and these are summed up to generate 1 equivalent pixel for that entire 3D subpart. It is then shifted to the next subpart depending on the value of the stride and the process is repeated



Example of 2D Convolution


Example of 3D Convolution

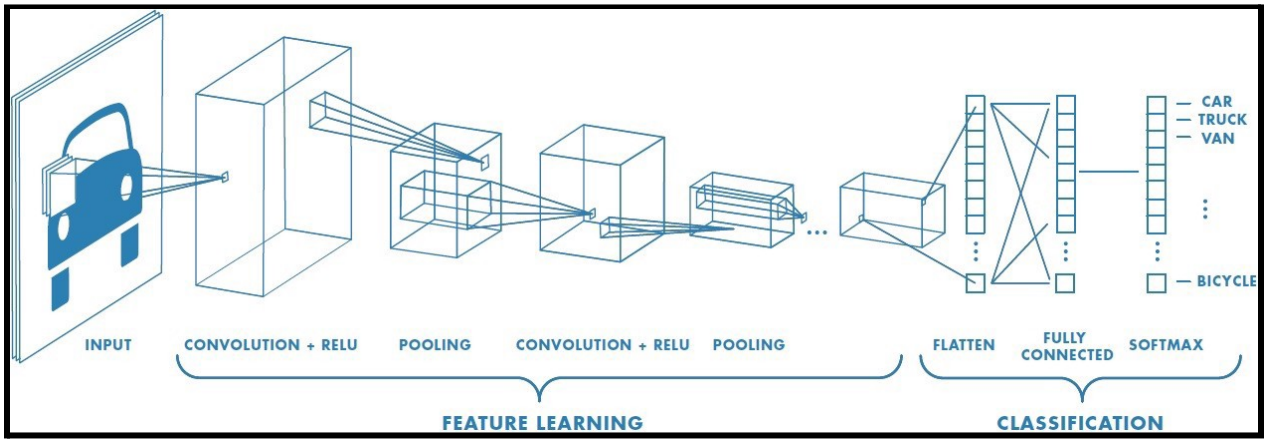
So, instead of applying individual weights to each pixel of the image, we are taking a small weighting matrix (referred to as a filter kernel) and then sliding it across the entire image. So, going back to our previous example of 1000 by 1000 by 3 image, if we use a 10 by 10 by 3 filter kernel, then the number of learnable weights is only 10*10*3 = 300, which is a vast reduction compared to the original case.

Development of CNN using Convolutional Layers:

With the reduction in learnable parameters, however, the feasibility of learning also decreases. So, in modern CNNs, a single layer contains many filter kernels, each of which operates on the input image, and the output of each filter is stacked to form the output 3D matrix. After this convolution operation, standard non-linear operations such as ReLu or the Sigmoid function are applied to each pixel of the output image. This entire process of convolution and then applying a nonlinear function is referred to as a convolutional layer. Modern CNNs are made up of stacking multiple convolutional layers one after the other.

Another useful operation used in some convolution layers is called Pooling. Pooling is of two types: Max Pool and Average Pool. It operates in the same way as to filter kernels but it operates separately on each channel. Max Pool replaces each subpart of each channel with the maximum pixel value in that subpart and Average Pool replaces each subpart of each channel with the average pixel value in that subpart. In this way, the Z dimension (number of channels) is preserved but the X and Y dimensions are significantly reduced. This operation is useful for reducing the size of features after each layer. Max Pool is generally used to preserve features in the original image, whereas, Average Pool is useful when we want to reduce noise in the original image.

Once the dimensions of the image have been reduced by a large factor after passing through multiple convolution layers, then the image can be unrolled (basically, flattened out) and then fed to standard Deep Neural Networks. This is the entire basic model structure of general CNNs.

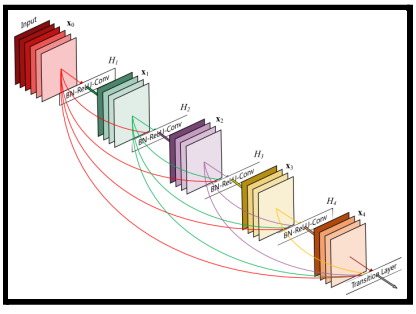


It resembles the way our brain interprets images. The initial convolution layers will extract basic features from the image, such as straight lines. The later layers will use these extracted features to extract more complex shapes such as circles and triangles. This process will continue with each layer learning to extract a more complex shape using the input from the previous layer. Each filter kernel of each layer will learn to extract a certain feature from the input image. This is exactly the process as followed by our brain as well.

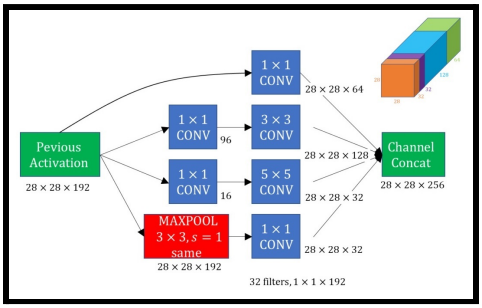
Some Specific Examples of CNN Models and their Applications:

Residual Networks: Due to such a large number of layers, weights can suffer from the Vanishing Gradient problem. One such modification is known as Residual Networks, in which each convolutional layer is connected to the next layer as well as to a much further layer in the model. In this way, the weights of that layer are updated not just by gradients of the next layer but also by gradients of the further layer which are not that diminished.

Inception Networks: One major hyper-parameter choice involved in the design of CNNs is the kernel dimensions in each layer. To solve this problem, the Inception Architecture has been proposed. Earlier, in a single convolutional layer, all the filters had the same dimensions, but in the Inception Architecture, each convolutional layer has multiple kernels, with each group of kernels having different dimensions, as well as Pooling kernels. The basic idea is that instead of needing to pick one of these filter sizes or pooling we want and committing to that, we can do them all and just concatenate all the outputs, and let the network learn whatever parameters it wants to use. The drawback is that the network complexity increases and much more time and data will be required for training.



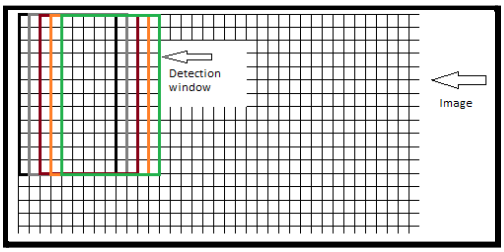
Residual Network



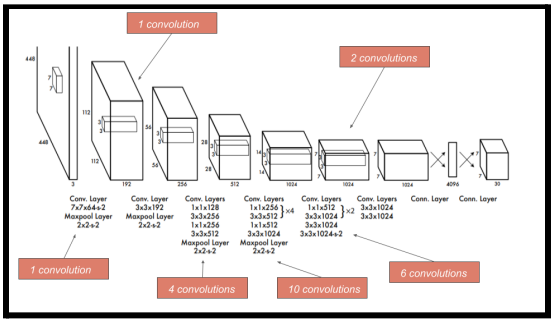
Single Layer of Inception Network

Object Detection and Localization using Sliding Window Algorithm: This method divides the input image into small sub-images and then passes each sub-image through the CNN. The CNN will tell us if an object is present in that window, and in this way, we will know the object location as well as the label. This is known as the "Sliding Window Algorithm" and can be implemented efficiently using convolution. However, the problem is that we need to fix the sizes of windows beforehand. One workaround is using windows of multiple sizes and repeatedly passing them through the CNN. The main problem is the tradeoff between accuracy and computational cost. If we decrease the stride length, then we will look at many windows inside the image and hence, our accuracy will increase at the expense of a very large computation cost. If we increase stride granularity, we might miss objects present in between 2 windows leading to loss of accuracy but lower computation cost.

Object Detection and Localization using Yolo Model: Another solution is that instead of just predicting a label for the image, it needs to predict the label as well as the bounding box dimensions for the object. Hence, the output vector for this task will be (n+4), where n = number of classes and each of these n units will output the probability of that class, and the remaining 4 units will give the Bounding Box centre X and Y coordinates and the height and width of the bounding box. The goodness of the predicted Bounding box is calculated using the Intersection over the Union (IoU) measure with the actual bounding box. The metric is similar to the Jaccard Similarity metric. It calculates the ratio of the intersection area to the union area of two bounding boxes. Another method called Non-Max Suppression is used to improve the performance by discarding low confidence bounding boxes. Anchor boxes can also be used to bias the shapes of the predicted bounding boxes to better match the object dimensions. The YOLO (You Only Look Once) model combines all of these modifications discussed above and is one of the most effective object detection algorithms.

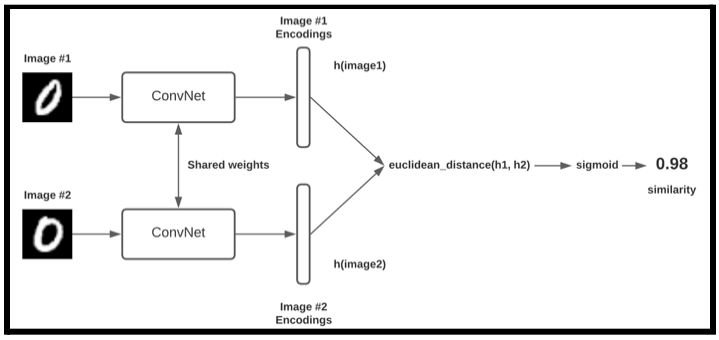


Sliding Windows Approach to Object Localization



YOLO Model

Facial recognition using Siamese Networks: The final application of CNNs to be discussed in this report is Facial recognition. It is a difficult problem, because, usually we don't have multiple images of the face of a single person available to us for training, and also, designing individual CNNs for each person is not feasible. However, there is a very elegant solution to this problem. As discussed above, the later layers in a CNN extract very high-level features of the input image. So, we can take a pre-trained general CNN model and then pass the original image of the face and the image to be classified through it. If both of them belong to the same person, then the extracted features by the later layers. So we can just compute the distance between the two extracted features and if the distance is small, then both images belong to the same person. This is known as a Siamese Network.



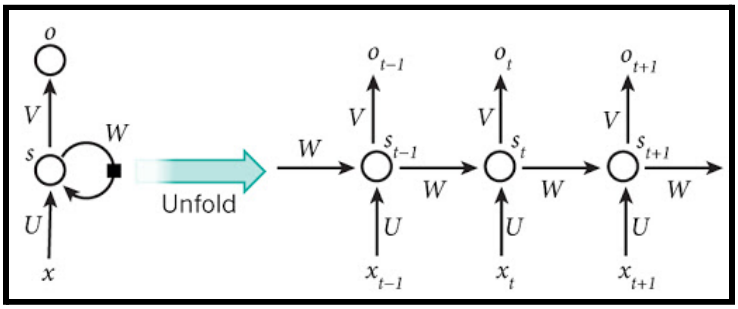
Sequential Neural Networks

Need for Sequential Models:

Conventional Neural Networks have limited success in dealing with sequential data. For processing sequential data, the model needed to retain some information about the previous states for better detection performance at each step. This previously could not be achieved using feed-forward and Convolutional Neural Networks. Another key issue was that sources of sequential data, such as text and speech produced samples having variable lengths. So, it was also desired that sequence models be capable of dealing with inputs of varying lengths, which was impossible with conventional neural networks.

Introduction to RNNs:

Recurrent neural networks(RNN) were based on David Rumelhart's work in 1986. Recurrent Neural Nets can be thought of as a generalisation of feed-forward neural networks, where the combined input to the model at each step includes the input and a latent state that encodes the sequence up to the current time-step. This latent state is updated at each step. The latent state obtained after processing the last element of the sequence is considered as encoding for the complete sequence and frequently used for text/document classification tasks. To elaborate on the operation performed at each step, the input and latent state from the previous steps are passed through single-layer neural networks. The outputs are combined and passed through another neural network and activation function to obtain the latent state at the current time-step (and a separate output state as well, as and if required by the target application). Some common applications of RNN include - POS tagging, Sentiment Classification, Question Answering, Speech Recognition, Machine Translation etc.



RNNs, although pretty effective, suffer from a few major problems. RNN is trained using the backpropagation-through-time algorithm, where the gradients from all future time-steps are accumulated for performing the weight update. Thus, the gradient at each step varies exponentially with the neural network weights. This can lead to the problem of exploding and vanishing gradients for long sequences, where the gradient can explode or vanish for large and small values of the weight magnitude respectively. While the first can lead to extremely large updates possibly leading to nan weight values, the second can lead to losing the gradient signals for longer sequences. Another key issue with conventional RNNs is the preference for sequential recency over syntactical recency which renders the model incapable of learning long-term dependencies.

Key RNN Variants:

LSTM: Long short-term memory (LSTM) networks, a special kind of RNNs were invented by Hochreiter and Schmidhuber in 1997. For incorporating long term dependencies, gates are used which can assume the values 0-1. Depending on the value of gates, information is written/read/erased from the LSTM latent and cell states. The gates included are Forget gate, Input gate and the Output gate, all of which are calculated at each step using the latent state

from the previous steps and the current input. The forget and input gates are used to calculate the current cell state from the past cell state and the cell content at the current step. That in turn is used along with the output gate to calculate the current latent state. If all forget gates are set to 1, the original cell state is retained for a long time. Thus, this method fixes the vanishing gradient problem.

GRU: A simple alternative to LSTM, GRU was introduced in Kyunghyun Cho in 2014. GRU used two gates, the Update gate and Reset gate and no separate cell state. The Reset Gate controls what part of the previous latent state is used to calculate the latent state content. The update gate controls what parts of the hidden states are updated/preserved. For update gate values close to 1, the hidden state is updated using the current hidden content and for values close to 0, the previous latent state is retained. GRU achieved similar performance to LSTM while greatly reducing space requirement.

Special RNNs: A key improvement over conventional RNN is the **Bidirectional RNN**. It can be thought of as two separate RNNs, where one processes the sequence in reverse. The hidden state at each step is obtained as the concatenation of the latent states from both RNNs. Bi-RNNs are frequently used with LSTMs and GRUs as well. However, this variant of RNN assumes the availability of the full context and thus will be ineffective in tasks such as language modelling, where the right-sided context is absent. Another key RNN variant is the **Multilayer or Stacked RNN**. This variant also includes multiple RNNs, where the hidden states from one RNN-layer are input to the next RNN-layer. Thus, this RNN variant resembles a large feed-forward neural network. Skip connections are frequently used as well and have shown good results. However, since the computation is sequential, they are mostly limited to a maximum of 4-6 layers.

Applications of sequence models:

RNNs find the most applications in language, speech and video processing. The field of Natural Language Processing(NLP) was revolutionised with the advent of RNNs. Some key applications in NLP include Named Entity Recognition, POS tagging(examples of word-level classification tasks) and sentiment classification, offensive message classification(examples of sentence-level classification tasks), Question Answering and Neural machine translation(examples of sequence-to-sequence tasks) and Language Modelling. RNNs also find extensive applications in speech processing and synthesis.

Further Developments:

In sequence to sequence tasks, encoding the source sentence in a single vector might not capture all the information. Considering the simple example of a machine translation task, the first output from the decoder will ideally be the translation of the first word in the input sentence. However, any information regarding that word might be diminished/lost. To fix this problem, the concept of the **Attention mechanism** was introduced. In this technique, the encoder hidden states at each step are stored, and a weighted sum is fed into the autoregressive decoder at each decoding step in addition to the previous time-step output. Any encoder state getting high attention(large weight) means that it is the most important for current word prediction. Thus, using attention, long-term dependencies can be effectively modelled.

Another key problem was the high computation requirement for sequence models. Sequential computation allows minimal parallelisation, thus limiting the network depth that we can afford to use. A breakthrough came with **Transformer** architecture, introduced in “Attention is all you need” authored by Vaswani et al in 2017. It removed the sequential computation altogether and instead focussed entirely on the attention mechanism. This allowed massive parallelisation, paving the way to OpenAI’s massive **GPT-3** language model. In 2019, Google researchers combined Transformers with ideas from self-supervised language pretraining and proposed the **BERT** model, variants of which are still the state-of-the-art in almost all language-specific tasks.