Bantaba Project

Introduction to Convolutional Neural Network using TensorFlow

Acknowledgements

I am very glad for having given this opportunity to work on this project. I am grateful for my Professor Nicholas Brown, teaching assistants Manogna Mantripragada, Nikunj Lad, Akshay Patel, my peers and my friends for the constant support and aid with the technical understanding of the concepts.

-Indupriya Kompi Sadasivappa

https://www.linkedin.com/in/indupriya-kompi-sadasivappa/

kompisadasivappa.i@husky.neu.edu

Table of contents

16. Reference

1. Overview

Artificial Intelligence (AI) has been witnessing a remarkable growth in bridging the gap between the capabilities of humans and machines. Artificial Neural Networks (ANN) have flourished in recent times in the processing of unstructured data, especially images, text, audio, and speech.

Convolutional neural networks (CNNs) work best for such unstructured data. Convolution Neural Networks in essence are neural networks that employ the convolution operation (instead of a fully connected layer) as one of its layers.

The agenda of this article is to enable the reader to understand the concepts of CNNs, applications of CNNs. A couple of examples deployed on jupyter notebook are also provided herewith this article to provides the insights on working on CNN.

What We See

What Computers See

2. Introduction

A **Convolutional Neural Network also knows as ConvNet** is a very widely used Machine Learning technique or algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods require filters are

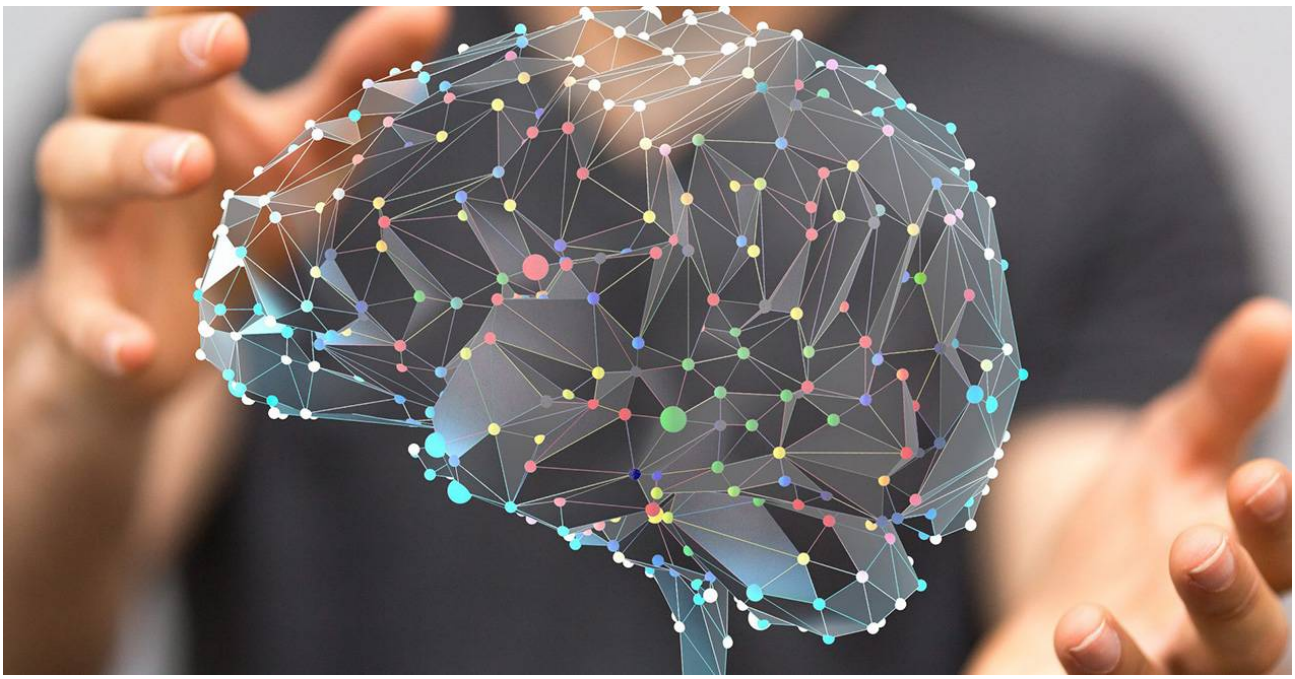
hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. CNNs are an incredibly successful technology that has been applied to problems wherein the input data on which predictions are to be made has a known grid like topology. Whenever there is a topology associated with the data, convolutional neural networks do a good job of extracting the important features out of the data.

From an architectural perspective, CNNs are inspired by multi-layer perceptron, the architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

A few of the popular MNCs that use Neural Nets for business purposes

The chief keystone element of convolutional neural networks is the processing of data through the convolution operation. Convolution of any signal with another signal produces a third signal that may reveal more information about the signal than the original signal itself.



Let’s go dive into details convolution neural networks!

---

3. History and Evolution of Convolutional Neural Network

CNN history begins with the neurobiological experiments conducted by Hubel and Wiesel (1959, 1962). Their work provided a platform for many cognitive models, and CNN replaced almost all of these. Over the decades, different efforts have been carried out to improve the performance of CNNs. CNNs have been applied to visual tasks since the late 1980s. In 1989, Yann LeCun proposed the first multilayered CNN and named ConvNet. Also, he proposed a supervised training of ConvNet using back propagation algorithm.

Different improvements in CNN architecture have been made from 1989 to date. As regards the architectural advancement of CNNs, recently, the focus of research has been on designing of new blocks that can boost network representation by exploiting feature-maps or manipulating input representation by adding artificial channels. Moreover, along with this, the trend is towards the design of lightweight architectures without compromising the performance to make CNN applicable for resource constraint hardware. Improvements came from the restructuring of processing units and the designing of new blocks. Most of the innovations in CNN architectures have been made in relation to depth and spatial exploitation. Depending upon the type of architectural modifications, CNNs can be broadly

categorized into seven different classes, namely; spatial exploitation, depth, multi-path, width, feature-map exploitation, channel boosting, and attention-based CNNs.

---

4. What is Convolution in Convolution Neural Network?

Convolution is the very first step that CNN undergoes. Convolution is purely a mathematical term. It performs an operation on two functions (f and g) to produce a new function that represents how the shape of one is modified by another function. Mathematically, convolution is defined as the integral product of two functions after one is reversed and shifted.

In the formula below, f and g represent the two input functions on which the convolution operation (*) is performed.

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau.$$

Convolution Operation

The role convolution operation in CNN is the very first step and plays a very significant role. It is further explained in detail in the **section 5.2** under ***Convolution Layer***.

---

5. Components of Convolutional Neural Network

A CNN typically consists four layers and three most important components in the network. They are,

5.1 Input Layer

5.2 Convolution Layer

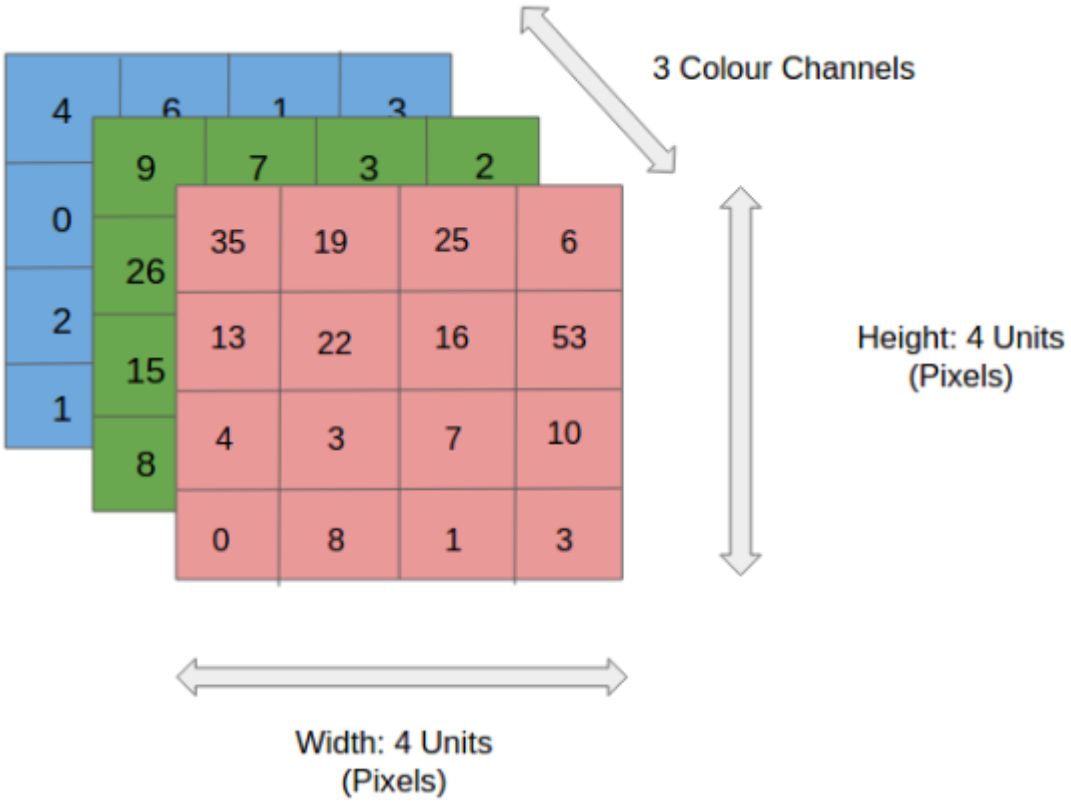
5.3 Pooling Layer

5.4 Fully Connected Layer

5.5 Activation Function

5.1 Input Layer

The input to this layer are images. It holds the pixels intensity of the image. Primarily, the images are fed in batches as four-dimensional tensors where the first dimension is specific to the image index, second and third dimensions are specific to the height and width of the image, and the fourth dimension corresponds to the different channels. For a colored image, generally we have the Red (R), Green (G), and Blue (B) channels, while for grayscale images we have only one channel. For instance, an input image with width 16, height 16 and depth 3 for RGB channels would have input dimensions 16x16x3



An RGB image represented in the form of matrix of pixels

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

The number of images in a batch would be determined by the mini-batch size chosen for the mini-batch stochastic gradient descent. The batch size is one for stochastic gradient descent. The inputs can be fed to the input layer in mini batches through TensorFlow placeholder tf.placeholder at runtime.

5.2 Convolution Layer

Convolutional Layer is the heart of any CNN network. In section 4, we discussed about the convolution operation. In this section we will study how it is implemented on the inputs of the CNN. The two functions are f and g that undergo this operation. One of the functions is the input to the network and the other function is called filter of certain kernel size. Input layer convolves with the filter to create an output called ***feature maps***. The number of output feature maps is equal to the specified number of filters. TensorFlow supports both 2D and 3D convolutions. However, 2D convolutions are widely used since 3D convolutions are computationally expensive and computationally memory intensive.

2D convolution happens along the spatial dimensions, while there is no convolution along the depth channel of the image volume. These filters help to detect features in the images. The deeper the convolutional layer is in the network, the more complicated features it learns. For instance, the initial convolutional layer might learn to detect edges in an image, while the second convolutional layer might learn to connect the edges to form geometric shapes such as circles and triangles. The even deeper convolutional layers might learn to detect more complicated features; for example, in the image classification of Cat versus Dog it might learn to detect eyes, nose, or other body parts of the animals.

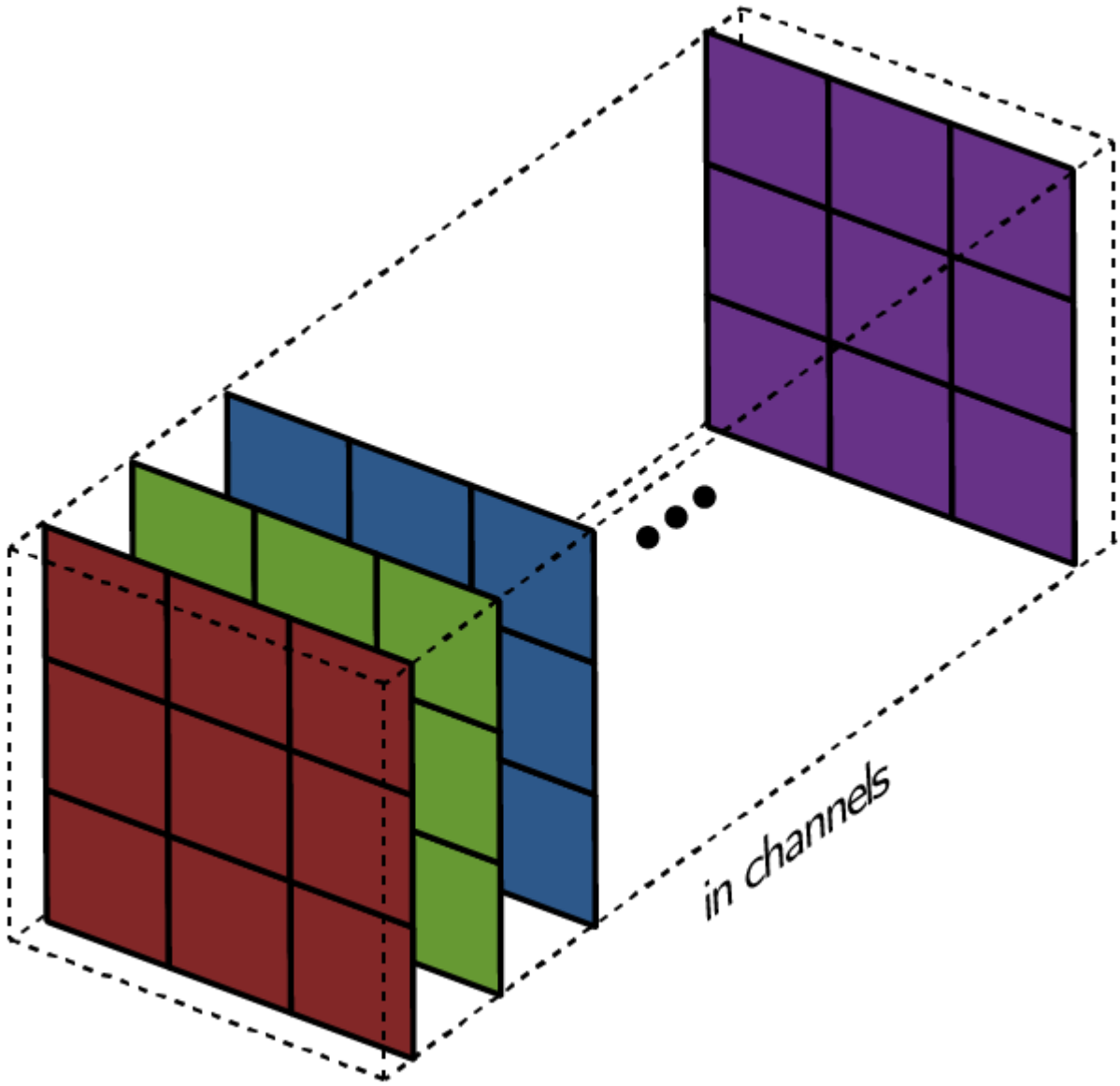
In a CNN, only the size of the filters is specified; the weights are initialized to arbitrary values before the start of training. The weights of the filters are learned through the CNN training process.

Following are the terms that helps in understanding the convolution layer more clearly:

5.2.1 Filter

A filter is a collection on kernels. Let us understand the term kernel by a very simple example. Let’s say you have to paint a wall or a piece of paper. Now, you’d start painting from one side and slide the painting brush across the entire area of wall/paper. In Machine Learning terms, this wall/paper is the input image and the painting brush is the filter. And

the region that is painted over is called receptive field. The filter is also an array of numbers. Filter size defines the height and width of the filter kernel. A filter kernel of gird size 3x3 would have nine weights. As the filter is sliding, or **convolving**, around the input image, it is multiplying the values in the filter with the original pixel values of the image. These multiplications are all summed up giving rise to a single valued number.

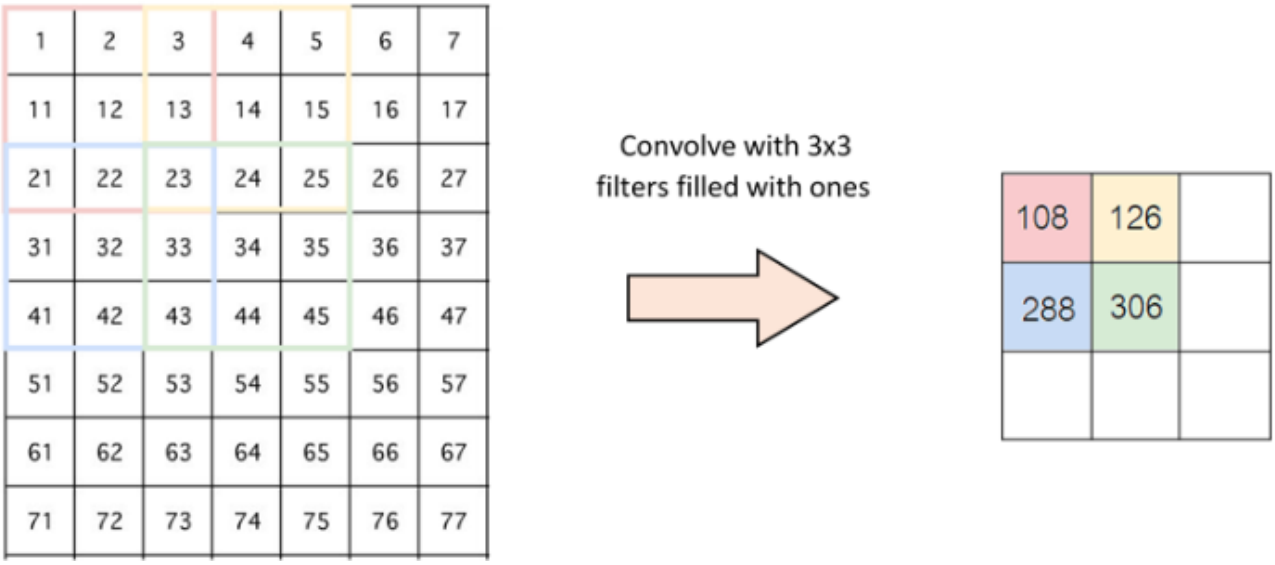


A filter-collection of kernels

5.2.2 Stride

We know that input image to the machine is the matrix of numbers called, pixels. The filters convolve with these pixels of input images and slides across. Strides determine by how many pixels the filter should move in the direction of convolution operation. In most of the cases, pixels are not skipped so as to not lose the information. However, in some cases a

stride of two is chosen along the height and width of the image. The resulting feature map will approximately be 1/4th of the original size of the image.

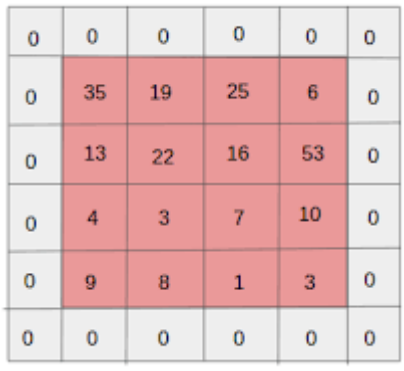


Strides

5.2.3 Padding

When we convolve input image with a filter, the size of the resultant that is, the output feature maps will be smaller than the input image. In order to preserve the size of the input in the output feature maps, the input image is appended with zeros on the border. Thus, padding is used for two purposes:

1. Preventing output from shrinking
2. Prevent losing information on the corner of the image



An input matrix of pixels with padding

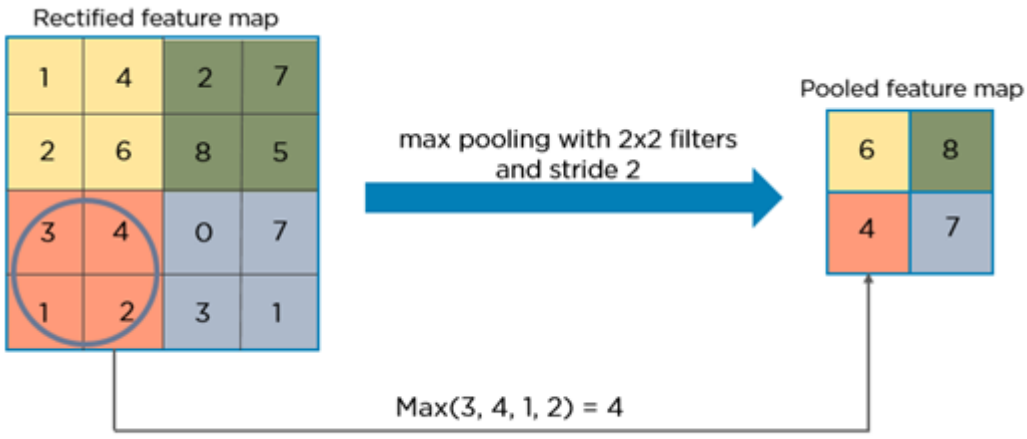
In TensorFlow, padding can be chosen either as “SAME” or “VALID”. Same ensures that the size of the input is retained in the output image while valid do not use padding.

The 2D convolution layer representation in keras

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=**None**, dilation_rate=(1, 1), activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

5.3 Pooling Layer

Pooling or down sampling operation is the next important operation which is followed by convolution. This operation summarizes the locality of the image. It progressively reduces the spatial size of the representation so as to reduce the amount of parameters and computations in the network. MaxPooling and AveragePooling are the two widely used pooling operations in TensorFlow. In maxpooling, the maximum pixel intensity of a locality is taken as the representative of the locality. In average pooling, the average of the pixel intensities around the locality is taken as a representative of that locality. The figure below represents the MaxPooling and Keras representation of maxpoolingD



Pooling operation-explained

MaxPooling2D [source]

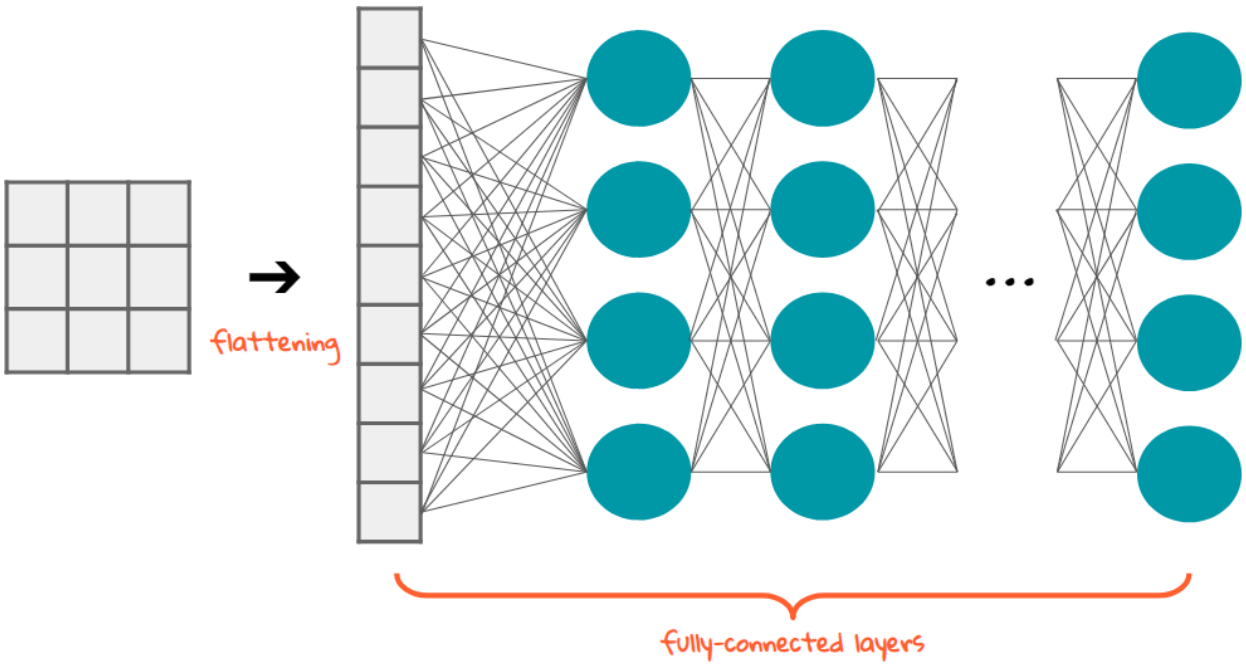
```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

Max pooling operation for spatial data.

MaxPooling in keras

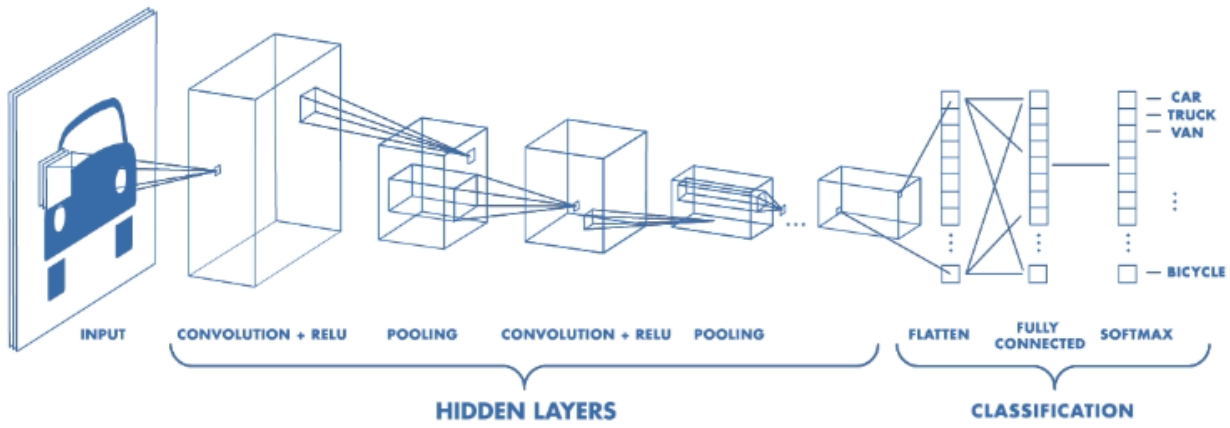
5.4 Fully Connected Layer

It is simply, feed forward neural network. They form the last few layers in the network. The input to this layer is the output from the final pooling/convolutional layer, which is flattened and then fed to the fully connected layer. Fully connected layers contain the traditional/conventional neurons that receive different sets of weights from the preceding layers; there is no weight sharing among them unlike the convolution operation. Each neuron in this layer will be connected to the all the neuron in the previous layer or to all the coordinate-wise outputs in the output maps through the separate weights.



Fully connected layer of CNN

With all the layers, integrated together, a typical convolutional neural network looks like what is shown below:



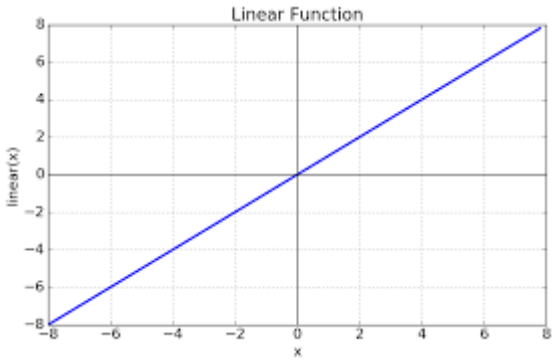
Fundamental Architecture of CNN

5.5 Activation Function and it's Types

A function that is used to get the output of node, also known as **Transfer Function**. In addition to this, activation function is also used to add some non-linearity into the function. Without activation function, the neural network could perform only non-linear mapping from input to the output. Activation functions maps the resulting output from 0 to +1 to -1. There are two types of activation functions namely,

5.5.1 Linear Activation Function

The output is linear and is not confined between any range.



A graph showing Linear function

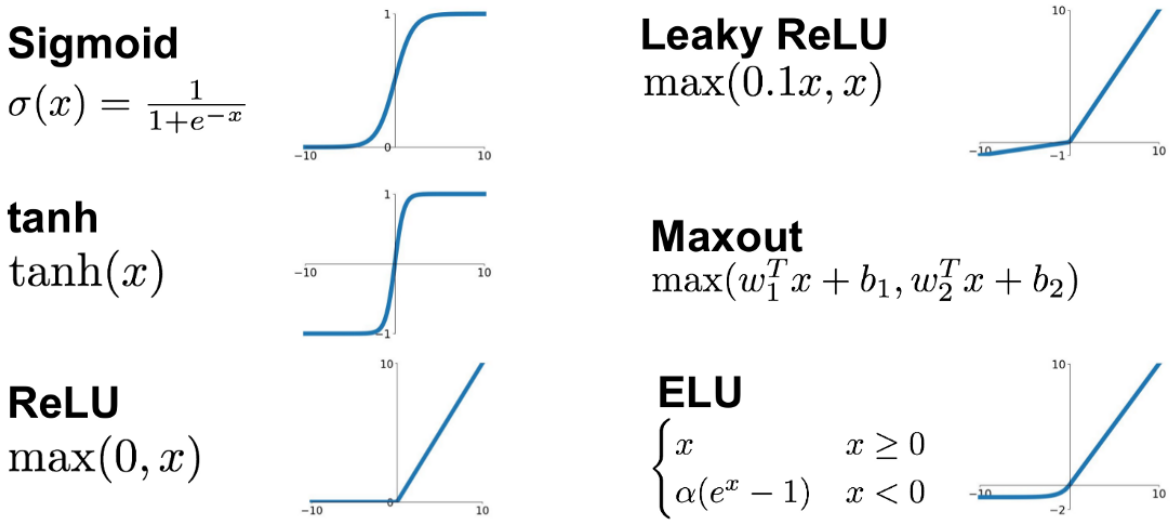
Input: f(x)

Range: (-infinity to +infinity)

5.5.2 Non-Linear Activation Function

These are the most widely used activation functions. The function does some type of transform the sum to a number that is often times between some lower limit and some upper limit. There are a few classifications of non-linear functions that are made based on the curves.

- Sigmoid or Logistic Activation Function
- Tanh or Hyperbolic Tangent Activation Function
- Rectifies Linear Unit (ReLU) Activation Function
- Leaky ReLU
- Exponention Linear Unit Activation Function
- Maxout Activation Function



Classification of non-linear activation function

---

6. Hyper-parameters in CNN

Hyper-parameters are the variables which determine the network structure and the variables that determine how the network is trained.

Hyper-parameters are **set before training** (before optimizing the weights and bias).

Neural networks can have many hyperparameters, including those which specify the structure of the network itself and those which determine how the network is trained.

6.1 Mini-Batch Gradient Descent Hyperparameters

1. Learning Rate: Low learning rate slows down the learning process but converges smoothly. High Learning rate speeds up the learning process but may not converge. Decaying Learning Rate is preferred.
2. Loss Function: Loss is used to calculate the gradients which are then used to update weights of the Neural Net. Keras and TensorFlow have various inbuilt loss functions for different objectives. They are

- Mean Squared Error (**MSE)** loss is used for regression tasks. As the name suggests, this loss is calculated by taking the mean of squared differences between actual(target) and predicted values.
- Binary Crossentrophy (**BCE)** loss is used for the binary classification tasks. Here, one output node classifies the data into two classes. The output value should be passed through a sigmoid activation function and the range of output is (0 – 1).
- Categorical Crossentrophy (CC) It is used with multi-class classification task. The number of output nodes is same as that of the number of class.
- Sparse Categorical Crossentrophy (SCC) It is same as CCE except that it does not require one-hot encoding of the target vector.

1. Batch-Size: The hyperparameter is the number of samples to work through the network before the weights are updated. The number of epochs is the number of complete passes through the training dataset.
2. Number of Training Iteration(T) is set using the principle of early stopping. Early stopping simply stops training once performance on a held-out validation set stops increasing. This is one of the best ways to prevent overfitting.
3. Momentum is a simple technique that improves both training speed and accuracy. Training a neural network is the process of finding value for the weights and biases so that for a given input values, the outputs match the target values.

6.2 Model Hyperparameter

The structure of the neural network itself involves numerous hyperparameters in its design, including the size and nonlinearity of each layer.

1. Number of Hidden Units: Using the same size for all hidden layers generally works better or the same as using a decreasing or increasing size. Plus, using a first hidden layer which is larger than the input layer tends to work better.
2. Weight Decay: When training neural networks, it is common to use "weight decay," where after each update, the weights are multiplied by a factor slightly less than 1. This prevents the weights from growing too large and can be seen as gradient descent on a quadratic regularization term.
3. Activation Sparsity: It may be advantageous for the hidden unit activations to be sparse. One way to enforce this is to use an L1 penalty (as discussed above) on the hidden unit parameters, provided that the activation has a saturating output around 0.
4. Non-Linearity: It is used to add the non-linearity in the function. More about this hyperparameter is discussed in the section **5.5.2**
5. Weight Initialization: Biases are typically initialized to 0, but weights must be initialized carefully. When initialized randomly, weights are often initialized in a range of [−r,r][−r,r] where rr is the inverse of the square root of the fan-in (sometimes added to the fan-out) of the unit.
6. Random Seeds and Model Averaging: It's common to train multiple models with multiple random seeds and use model averaging (bagging, Bayesian methods) to improve performance.

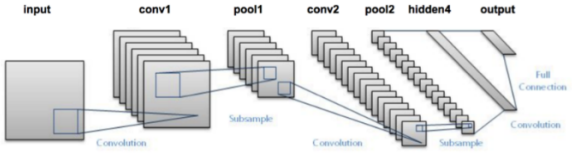
---

7. Different Architectures of CNN

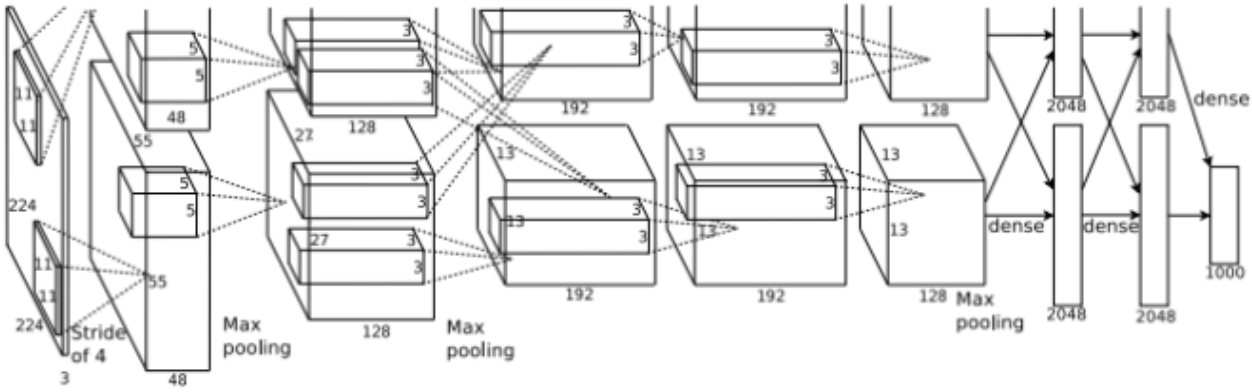
Different improvements in CNN architecture have been made from 1989 to date.

7.1 LeNet was proposed by LeCuN in 1998 classifying handwritten digits successfully for OCR-based activities such as reading ZIP codes, checks, and so on. It is famous due to its historical importance as it was the first CNN, which showed state-of-the-art performance on hand digit recognition tasks. It has the ability to classify digits without being affected by small distortions, rotation, and variation of position and scale. LeNet was the first CNN

architecture, which not only reduced the number of parameters but was able to learn features from raw pixels automatically.

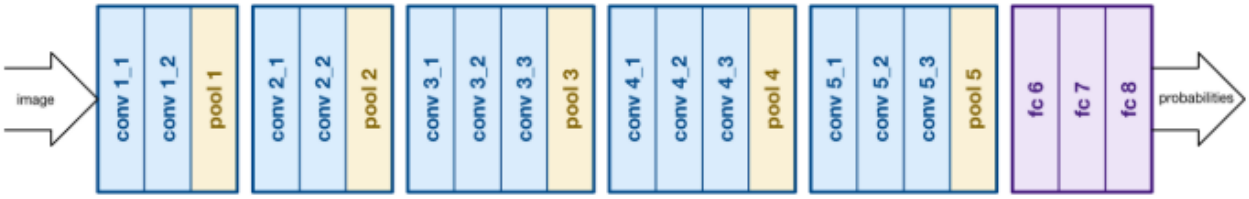


7.2 AlexNet AlexNet was proposed by Krizhevesky et al., which enhanced the learning capacity of the CNN by making it deeper and by applying several parameter optimizations strategies. AlexNet consists of five convolutional layers, max pooling layers, and dropout layers, and three fully connected layers in addition to the input and output layer of a thousand class units.



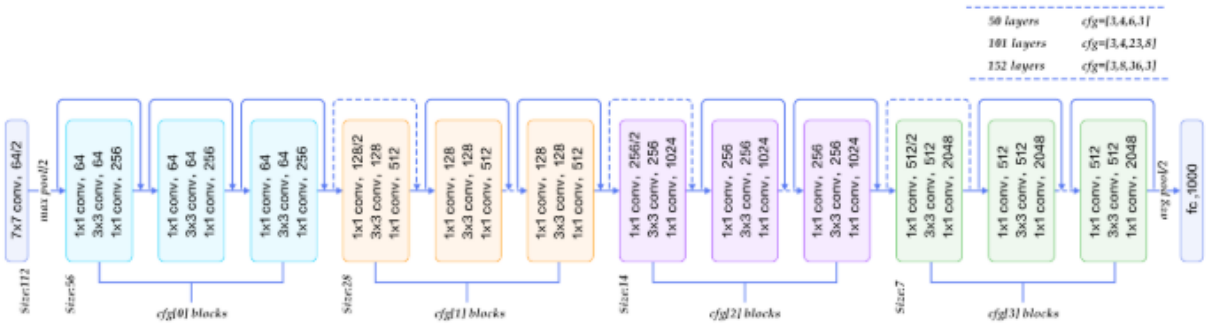
AlexNet Architecture

7.3 VGG16 In this regard, Simonyan et al. proposed a simple and effective design principle for CNN architectures. Their architecture, named as VGG, was modular in layers pattern. VGG was made 19 layers deep compared to AlexNet to simulate the relation of depth with the representational capacity of the network. VGG regulates the complexity of a network by placing 1x1 convolutions in between the convolutional layers, which, besides, learn a linear combination of the resultant feature-maps.



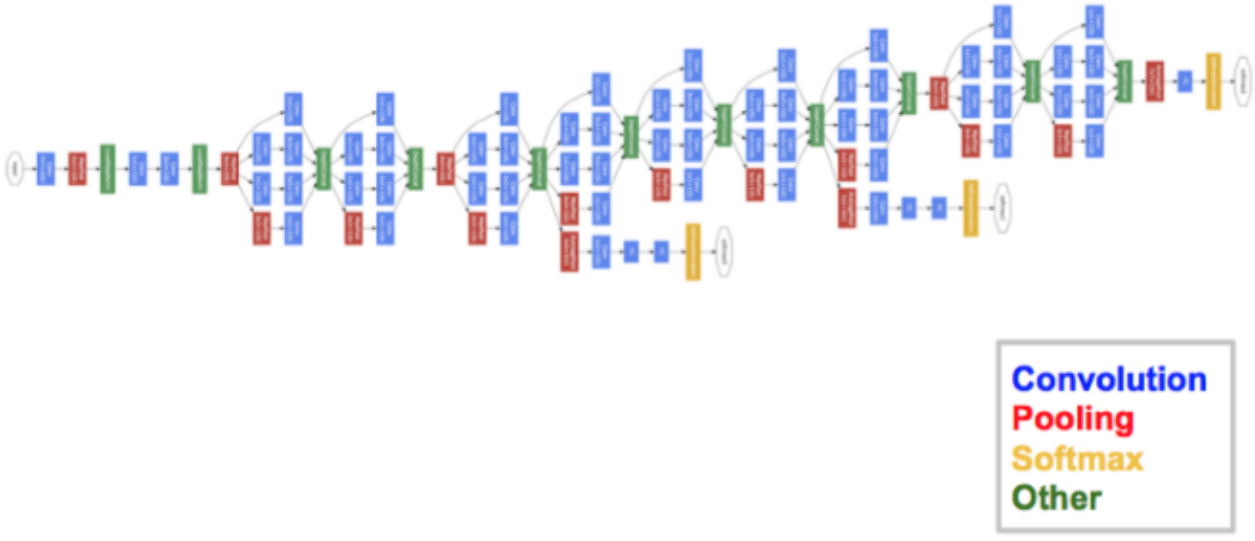
VGG16 Architecture

7.4 ResNet is a 152-layer-deep convolutional neural network from Microsoft. ResNet revolutionized the CNN architectural race by introducing the concept of residual learning in CNNs and devised an efficient methodology for the training of deep networks.



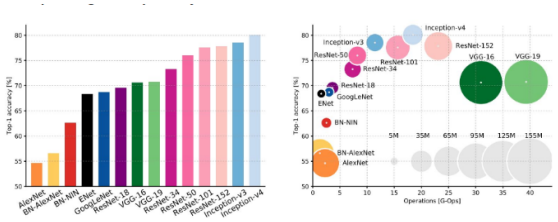
ResNet Architecture

7.5 GoogleNet GoogleNet architecture was to achieve high accuracy with a reduced computational cost. It introduced the new concept of inception block in CNN, whereby it incorporates multi-scale convolutional transformations using split, transform and merge idea. In GoogleNet, conventional convolutional layers are replaced in small blocks similar to the idea of substituting each layer with micro NN as proposed in Network in Network (NIN) architecture.



GoogleNet Architecture

Let’s have a look at the summary and analysis of the different architectures of CNN



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

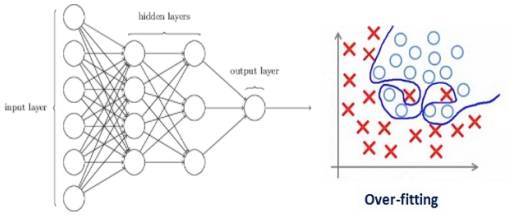
Summary Table

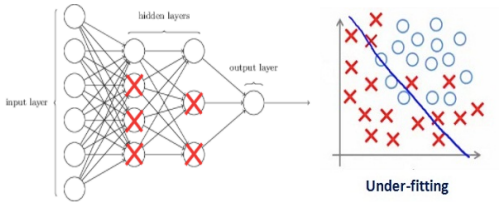
Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

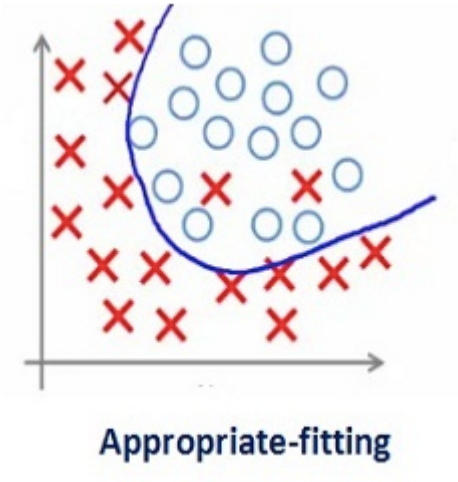
---

8. Regularization Techniques in CNN

Whenever a model is performing well on training data but not so-efficiently on the test data, we can say that the model is overfitting. A condition with high variance and cannot be relied upon. Thus, to overcome such situations or to avoid it, we use **Regularization.** It is a strategy or technique that is used to reduce the error on test data at an expense of increased training error.







Overfitting, underfitting and appropriate fitting

Different Regularization Techniques in Deep Learning are:

8.1 L2 & L1 Regularization Technique

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

In L2 Regularization, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$ Lambda is a regularization parameter, whose value is optimized for better results. L2 Regularization is also called as weight decay as it forces the weights to decay towards zero.

In L1 Regularization we've,

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||$$ here, the absolute values of weights are penalized.

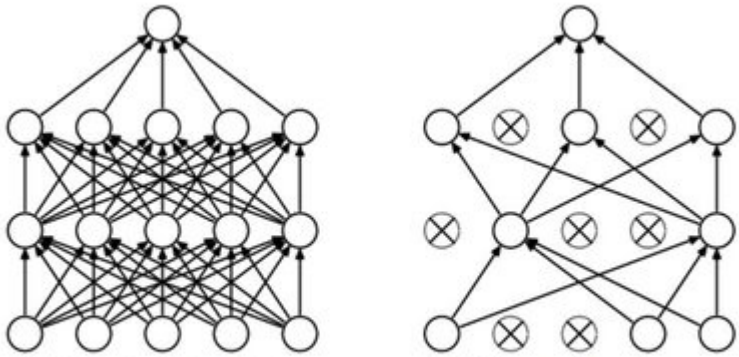
Keras Implementation:

```
from keras import backend as K

def l1_reg(weight_matrix):
    return 0.01 * K.sum(K.abs(weight_matrix))

model.add(Dense(64, input_dim=64,
                kernel_regularizer=l1_reg))
```

keras implementation of regularization

8.2 Dropout

It is computationally inexpensive but one of the most powerful techniques in regularization. In this technique, at every iteration, the weights of the neurons or nodes are dropped randomly i.e., the nodes are removed randomly along with their incoming and out-going connections.



Dropout, A regularization Technique

Keras Implementation:

```
from keras.layers.core import Dropout

model = Sequential([
 Dense(output_dim=hidden1_num_units, input_dim=input_num_units, activation='relu'),
 Dropout(0.25),

Dense(output_dim=output_num_units, input_dim=hidden5_num_units, activation='softmax'),
 ])
```

Keras implementaion of dropout

8.3 Data Augmentation

The output efficiency can be easily increased by training more and more data and more diverse data. While dealing with the image data, we use data augmentation techniques to introduce more diversity into the training data. It is used to zoom-in the images, flipping, rotating, scaling, shifting, etc

Operations performed using Data Augmentation

Keras Implementation:

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(horizontal flip=True)
datagen.fit(train)
```

Keras Implementation

8.4 Early Stopping

Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model. This is known as early stopping.

Graph showing early stopping

Keras Implementation

```
from keras.callbacks import EarlyStopping

EarlyStopping(monitor='val_err', patience=5)
```

keras implementation of keras function

8.5 Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. It aids stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. It smoothens the flow of gradient and acts as a regulating factor, which thus helps in improving the generalization of the network

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

batch normalization

---

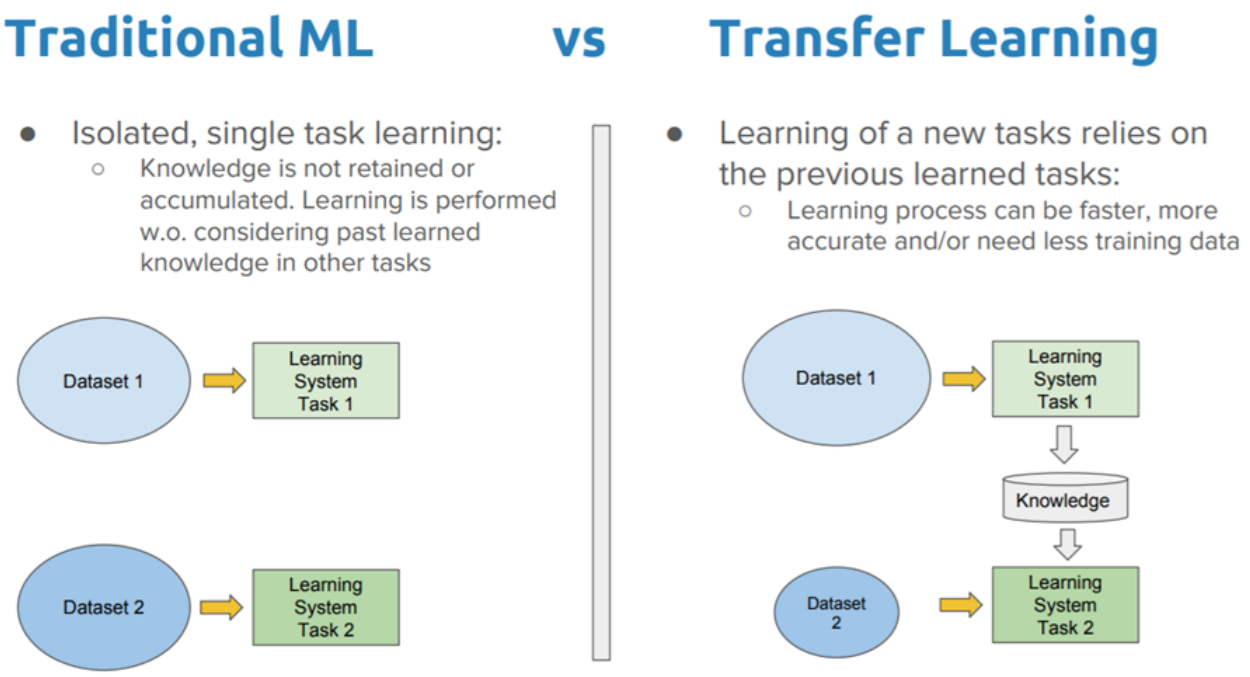
9. An Introduction to Transfer Learning

It is a popular approach in deep-learning where pre-trained models are used as a starting point of the second task. Let us understand the concept of Transfer Learning with an example:

Firstly, a big dataset also called base dataset is trained. It is also advisable to obtained pre-trained model for that dataset. Name it ‘Training Network’

Next we obtain a new dataset called target dataset which should be relatively smaller than the first one. Now train this network. This network shall have layers upto *nl.* The layer from 1 to *nk* (where *nk* < *nl*) are same as that of the networks from training network. The idea here is to inherit the features learned that are already learned in order to avoid the need to learn it again. The layers from *nk* to *nl* are used to extract the rest of the features required for further classification.

It is a very powerful method to avoid overfitting and underfitting. Generally, this method is used with the pre-trained models.



Difference between Traditional ML and TL

---

10. Implementation of Convolutional Neural Network using

Herewith attached is the link to my GitHub repository. It contains the jupyter notebook with the implementation of Convolutional Neural Network that is built using kannada digits dataset taken from kaggle. You can find the data set here: https://www.kaggle.com/c/Kannada-MNIST

https://github.com/IndupriyaKompi/BantabaProject

---

11. Applications of CNN

CNNs have been successfully applied to different ML related tasks, namely object detection, recognition, classification, regression, segmentation, etc. They generally need a large amount of data for learning. All of the areas mentioned earlier in which CNN has shown tremendous success have relatively sufficient labeled data, such as traffic sign recognition, segmentation of medical images, and the detection of faces, text, pedestrians,

and humans in natural images. Some of the interesting applications of CNN are discussed below.

1. CNN based computer vision and related applications
2. CNN based natural language processing
3. CNN based object detection and segmentation
4. CNN based image classification
5. CNN based speech recognition
6. CNN based video processing
7. CNN for resource limited systems
8. CNN for 1D-Data

---

12. Challenges of CNN

1. CNN is referred to as ‘Black box’. No matter how much we try to understand what is going on in the network, it is not possible to clearly interpret the network.
2. Training of CNN on noisy image data increases the misclassification error
3. Deep CNNs are based on supervised learning mechanisms, and therefore, the availability of large and annotated data is required for its proper learning
4. Hyper-parameter selection highly influences the performance of CNN, a little change in the hyper-parameter values can affect the overall performance of a CNN. That is why careful selection of hyper-parameters is a major design issue that needs to be addressed through some suitable optimization strategy.
5. CNN demands high performance hardware resources like GPU or TPUs.

---

13. Future Scope

Availability of multiple CNN architecture is a very promising in the field of research and in future, is more likely to be the most widely used AI techniques.

1. Ensembling of multiple learning techniques or combination of multiple diverse techniques can aid in improving the robustness of the model.

2. In the future, the concept of pipelining can be used to accelerate the training of large models and to scale the performance without tuning hyper-parameters. This can help in reducing the hardware capabilities.
3. Hyper-parameter tuning is a tedious and intuition driven task, which cannot be defined via explicit formulation. In this regard, Genetic algorithms can also be used to automatically optimize the hyper-parameters by performing searches both in a random fashion as well as by directing search by utilizing previous results.
4. New paradigms could be implemented that can enhance the learning capacity of CNN by incorporating informative feature-maps that can be learned using auxiliary learners at the intermediate stages of CNN

14. Conclusion

This chapter of Convolutional Neural Network serves as a guide to a beginner in the field of Deep Learning. We have covered major chuck of concepts in CNN, implementation of this network in Keras, TensorFlow. Along with this chapter is the attached ipynb file that clearly explains the implementation of CNN network using TensorFlow step-by-step. Majority of the concepts that are discussed here, you can see them implemented there in the notebook. It serves as an hands-on example. You may download it and play around with different hyperparameter.

Personally, I have learnt a lot of things and I believe there is always scope for improvement. CNN is a very vast subject by itself and very dense. With many innovations that are coming up, CNN is playing a vital role and has majorly contributed to today's world

15. About the author

Indupriya Kompi Sadasivappa is a currently enrolled master's student at Northeastern University, Boston. You may reach out here: kompisadasivappa.i@husky.neu.com

---

16. References

I have resourced the contents from multiple sources. From blogs, books, github repositories, youtube videos, kaggle kernels, discussion with friends, peers, my professors and teaching

assistants. Here are the links to the sources that I have referred to.

[1] https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

[2] https://cs231n.github.io/convolutional-networks/

[3] https://arxiv.org/abs/1603.07285

[4] https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1-convolution-operation

[5] https://thesai.org/Downloads/Volume10No6/Paper_38-Hyperparameter_Optimization_in_Convolutional_Neural_Network.p

[6] https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[7] https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniqu

[8] https://towardsdatascience.com/regularization-techniques-for-neural-networks-e55f295f2866

[9] Advanced Applied Deep Learning by Umberto Michelucci

[10] Pro Deep Learning with TensorFlow by Santanu Patanayak

[11] Convolutional Neural Network in Convolution in Xiaobo Huang

[12] A Survey of the Recent Architectures of Deep Convolutional Neural Networks by Asifullah Khan, Anabia Sohail, Umme Zahoora, Aqsa Saeed Qureshi.

[13] A few of Images are taken from google directly and a few more from websites mentioned above.