Artificial Neural Networks : An Elegance in Complexity.

By : Karthick Srinivas S. & Samuela Abigail Matthew.

Abstract:

This Paper presents a study done on the topic of Artificial Neural Networks (ANNs) by two students belonging to the Dept. of Artificial Intelligence & Data Science. The objective of the study is to essentially understand how to conceptualize, realize and build fundamental ANN architectures from the "ground-up".

This Paper concentrates on two classic neural network architectures: *1.)* Multi-Layer perceptrons (MLPs) 2*.)* Convolutional Neural Networks (CNNs). This paper will provide a thorough examination on various technical aspects associated with the aforementioned ANNs.

This Paper initially focuses on, the mathematical foundations and intuitions that enable the ANNs in consideration, to model complex, non – linear relationships and hence approximate functions efficiently. The paper then goes on to present practical implementations of the relevant ANNs on two popular Deep Learning frameworks – *Keras (ref. Chollet, 2015) & PyTorch* (ref. *Paszke and Gross, 2016*). The Practical implementations will be evaluated by using the – *Modified National Institute of Standards and Technology* (MNIST) Handwritten Digit Recognition Dataset (ref. *LeCun, Cortes and Burges, 1998)* and the *German Traffic Sign Recognition Benchmark* (GTSRB) dataset.

After the model evaluation, some conclusory results and inferences regarding the study will be presented following which, the paper will conclude.

<u>Multi-Layer Perceptrons – MLPs</u>

Introduction:

The *"neuron"* is the functional and structural unit of the entire human nervous system, more specifically the human brain. The brain is in fact a massive biological network of over 10,000 neurons (ref. *Buduma & Papa, 2022).* This *"massive network"* is responsible for enabling humans to perceive and experience the world around them. Deep Learning in its entirety is solely based on the philosophy of understanding this natural structure (the *neuron*) and using this understanding to build machine learning models that solve cognitive tasks in an analogous fashion.

Keeping in mind the working mechanism of the biological neuron, we can translate the fundamental understanding into an artificial model that can be represented on a computer. Such a model was first described in 1943 by Warren S. McCulloch and Walter H. Pitts in their paper, *"A Logical Calculus of the ideas Immanent in nervous activity"* (ref. McCulloch & Pitts, 1943).

It is very easy to observe that, just like a biological neuron, an artificial neuron also takes in some number of inputs (*x1, x2, ..., xn*). Each input is multiplied by a certain weight (*w1, w2, ..., wn*) and the weighted inputs are summed up to produce one *logit* of the neuron. In many cases the logit also includes a *bias* which is a constant value. The logit is then passed through a function *f* to produce the output *y = f(z).* The output is then transmitted to other "*member neurons"* of the neural network.

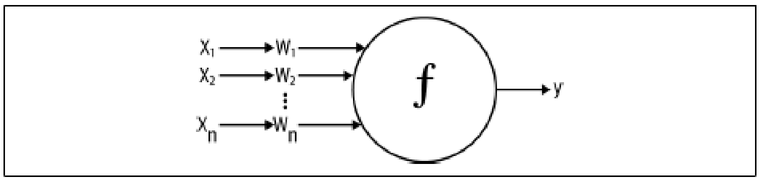


Figure 3-7. Schematic for a neuron in an artificial neural net

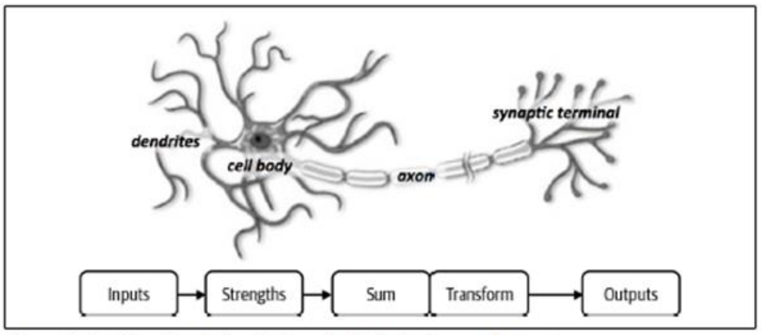


Figure 3-6. A functional description of a biological neuron's structure

So, in Summary, an artificial neuron is associated with two *vectors* as far as the input is concerned. A vector $X = [\ x1\ x2\ \ldots xn]$ usually represents the inputs and a vector

$W = [\ w1\ w2\ \ldots\ wn]$

Usually represents the associated weights. We can re-express the output of the neuron as -

$Y = f\,(x \cdot w + b)$ where, b is the *bias* term.

Linear Neurons and their limitations:

Most neurons are defined by the function f they apply their *logit z*. If we consider layers of neurons that use a linear function in the form of $f(z) = az + b$, we find that they are easy to compute with, but they run into serious limitations in a *multitude* of non-trivial cases. It can be showed that any artificial neural network consisting of linear neurons only can be expressed as a network with no hidden layers.

The Absence of hidden layers poses a problem as hidden layers enables the neural network to learn important *features* and other *complex* relationships from the data at hand. Therefore, in order to *circumvent* the above-mentioned flaw, we have to use neurons that employ some sort of a *non – linearity*.

Major Types of Non-Linear Neurons:

1.) **Sigmoid Neuron**: it uses a *non – linear* activation function on a logit such that the output *varies* as the size of the logit changes.
The obtained output will be closer to *zero* if the logit is a *small sized* one whereas the output will be closer to *one* if the logit is a *large sized* one. In between the two extremes the neuron assumes an *S-Shape*.

$$f(z) = \frac{1}{1 + e^{-z}}$$

The activation function is of the above indicated form.

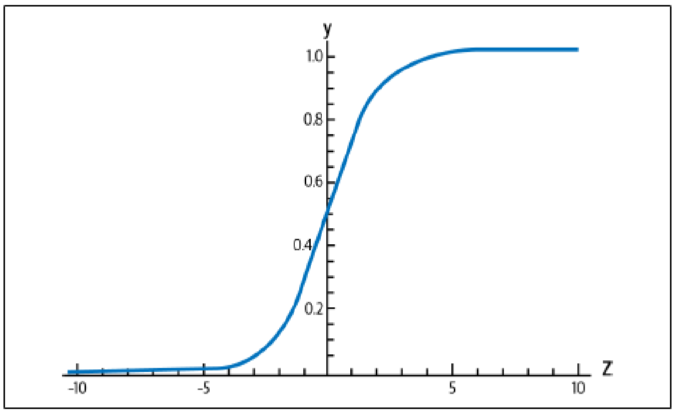


Figure 3-11. The output of a sigmoid neuron as z varies

2.) ***Tanh neurons***: They follow a similar kind of S – Shaped Non – Linearity but instead of values varying between 0 and 1, the values vary between -1 and 1. As expected, the activation function will be *f(z) = tanh(z).*

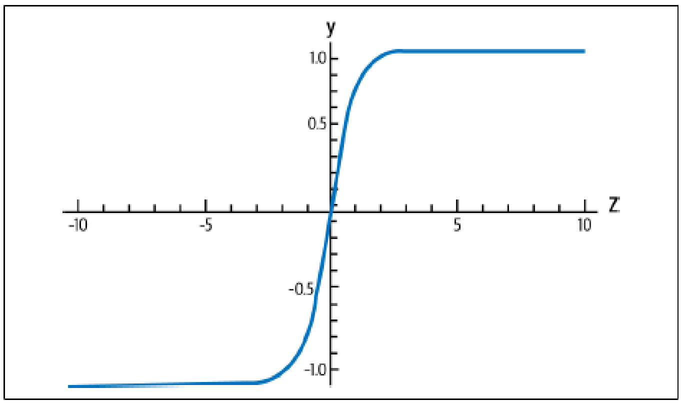


Figure 3-12. The output of a tanh neuron as z varies

3.) ***ReLU neuron***: These Neurons use an activation function which has a *"hockey-stick"* like shape. Despite some drawbacks this non-linear neuron has become the choice for a variety of applications in many fields of Artificial Intelligence. The activation function is of the *form f(z) = max(0, z).*

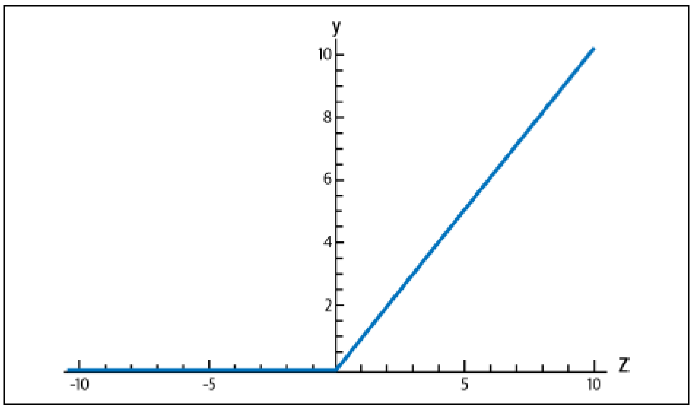


Figure 3-13. The output of a ReLU neuron as z varies

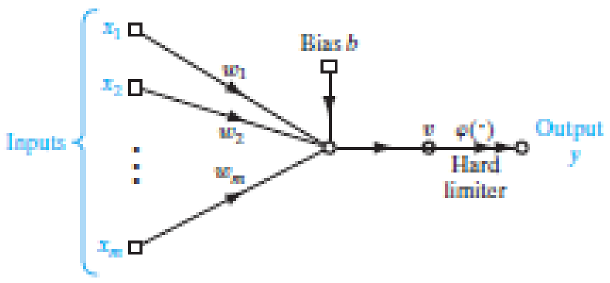
An Overview of Rosenblatt's Perceptron:

Rosenblatt's perceptron was built using a non-linear neuron – the *McCulloch & Pitts neuron*. It's neural modelling is in such a way that it is composed of a *linear combinator* along with a *hard limiter*. Accordingly, the neuron produces an output equal to +1 if the hard limiter input is positive and -1 if it is negative.

The Goal of Rosenblatt's perceptron is to *correctly classify* a set of externally applied inputs *x1, x2, ..., xn* into one of two classes, *c1 or c2*. The *decision rule* for the classification is to assign the point represented by the inputs *x1, x2, ..., xm* to class *c1* if the perceptron output *y* is *+1* and to class *c2* if it is *-1*.

The signal flow graph of the Rosenblatt's Perceptron is shown below:

FIGURE 1.1 Signal-flow graph of the perceptron.



Foundations of MLPs:

All the discussions presented earlier will serve as pre-requisite knowledge in introducing a very fundamental type of ANN Architecture called the Multi-Layer Perceptron (MLPs *in short*). Shown below is an MLP Architecture composed of 2 hidden layers.

In its general form, it is very easy to see that MLPs are *fully connected* networks. That is, a neuron at any layer of the network is connected to *all* the neurons in the previous layer. The signal flow through the network progresses in the *forward direction*, from left to right on a *layer-by-layer* basis.

There are two kinds of signals that are associated with the notion of MLPs:

1.) ***Function Signals*****:** A function signal is presumed to perform a *useful function* at the output end of the network. At each neuron of the network through which a function signal passes, the signal is calculated as a function of the inputs and the *associated weights* applied to that neuron. Function signals are also commonly known as *Input Signals*.

2.) ***Error Signals:*** An error signal originates at an output neuron of the network, and it propagates *backward* (layer by layer) through the network. We refer to it as an error signal because its computation involves an *error-dependent-function* in one form or another.

Now, let us have a *brief illustrative discussion* on the two types of computations performed by the neurons present in a neural network.

Each hidden or output neuron of an MLP is designed to perform two computations:

1.) The computation of the function signal appearing at the output of each neuron, which is expressed as a *continuous non-linear function* of the input-signal and *synaptic weights* associated with that neuron. This is done on the *forward-propagation* stage of the training process.

2.) The computations of an estimate of the *gradient vector* (i.e., the gradients of the error surface with respect to the *weights* connected to the inputs of a neuron). This function is computed during the *backward-propagation* stage of the training process.

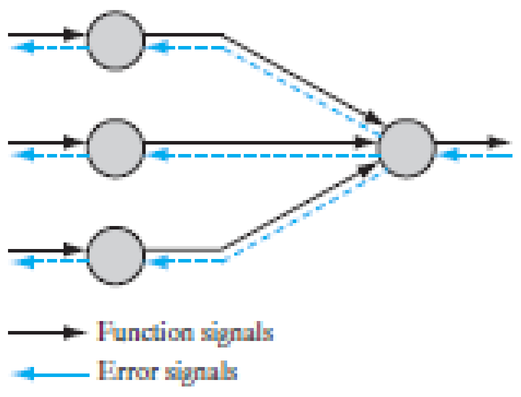


FIGURE 4.2 Illustration of the directions of two basic signal flows in a multilayer perceptron: forward propagation of function signals and back propagation of error signals.

An Overview of Hidden Neurons, Batch and On-line learning:

1.) The ***hidden neurons*** act as feature detectors, and hence they play a critical role in the operation of a multilayer perceptron. As the learning process progresses across the MLP, the hidden neurons begin to gradually *discover* its salient features that characterize the training data. They do so by performing a *non-linear transformation* on the input data into a new space called the *feature space*. It is easy to separate the *classes of interest* in a pattern classification task in the feature space.

2.) ***Batch Learning*****:** In the batch method of *supervised learning*, adjustments to the synaptic weights of the multilayer perceptron are performed *after* the presentation of *all* the *N* examples in the training sample *t* that constitute one *epoch* of training. Adjustments to the synaptic weights of the multilayer perceptron are made on an *epoch-by-epoch basis.*

3.) ***On-Line Learning:*** In the on-line method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are performed on an *example-by-example basis*. The cost function to be minimized is therefore the *total instantaneous error energy e(n).*

Training of MLPs:

- ***Backpropagation algorithm:*** it is essentially an algorithm that working with the "back-propagation" of *errors*. It is an algorithm that is designed to test for errors *working back* from output nodes to input nodes. It is hence an important *mathematical tool* for improving the accuracy of the predicted outputs in *Machine Learning* and *Deep Learning*.

Note:

1.) ANNs use Backpropagation as a learning algorithm to compute a gradient descent with respect to the weight values for the various inputs. By comparing desired outputs to achieved system outputs, the systems are tuned by adjusting the associated weights to narrow the difference between the two values as much as possible.

2.) Objective of Backpropagation: to use mathematical techniques like stochastic gradient descent (SGD) to *train* multilayer networks that *update weights* and *minimize loss.*

Advantages of Backporpagation:

1.) It does not have any parameters to tune except for the number of inputs.
2.) It is highly adaptable and efficient, and it does not require prior knowledge about the network.
3.) It does not need any special functions.

Disadvantages of Backporpagation:

1.) Prefers a matrix-based approach over a mini-batch approach.
2.) Performance is highly dependent on input data.
3.) Training is a time and resource intensive process.

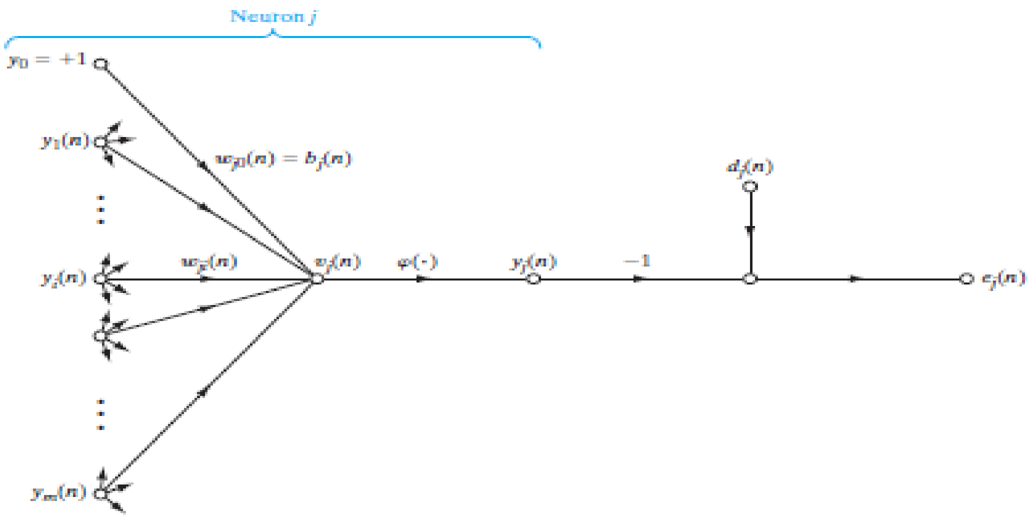


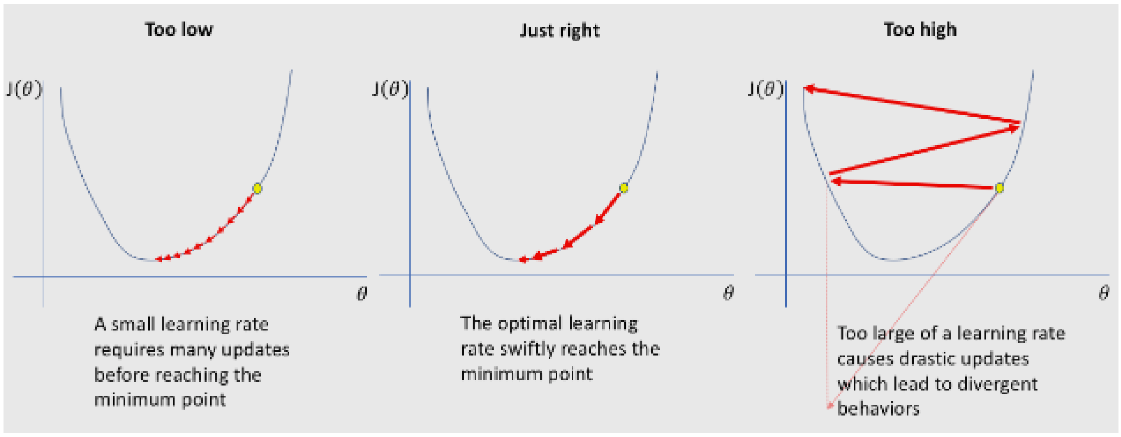
FIGURE 4.3 Signal-flow graph highlighting the details of output neuron j.

- ***Activation Function:*** The activation function or the *transfer function* plays an important role in obtaining an output from a node. It determines the output of a neural network as Boolean entities like *Yes or No*. It maps the resulting values in between *0 to 1 or -1 to 1* etc.

  The initial parts of the paper already present an elaborate discussion on activation functions – their *types*, their *graphical representations*, and their *applications.*

- ***Learning Rate:*** It is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

  The Learning rate of an algorithm must be optimally set in such a way that it does not lead to inefficiencies in terms of the updating of weights. A very low value of the learning rate requires many updates before reaching the minimum point while a learning rate too large causes drastic updates which leads to divergent behaviors.


Too low
Just right
Too high
$J(\theta)$
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

- ***Learning Termination:*** In general, the back-propagation algorithm cannot be shown to converge and there are no well-defined criteria for stopping its operation. However, there are some reasonable criteria for stopping its operation which may be useful in termination after weight adjustments.

  The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

  The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

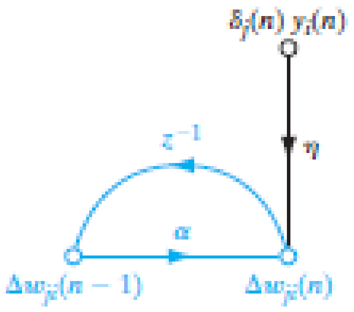


FIGURE 4.6 Signal-flow graph illustrating the effect of momentum constant α, which lies inside the feedback loop.

Convolution Neural Network for Traffic Sign Classification

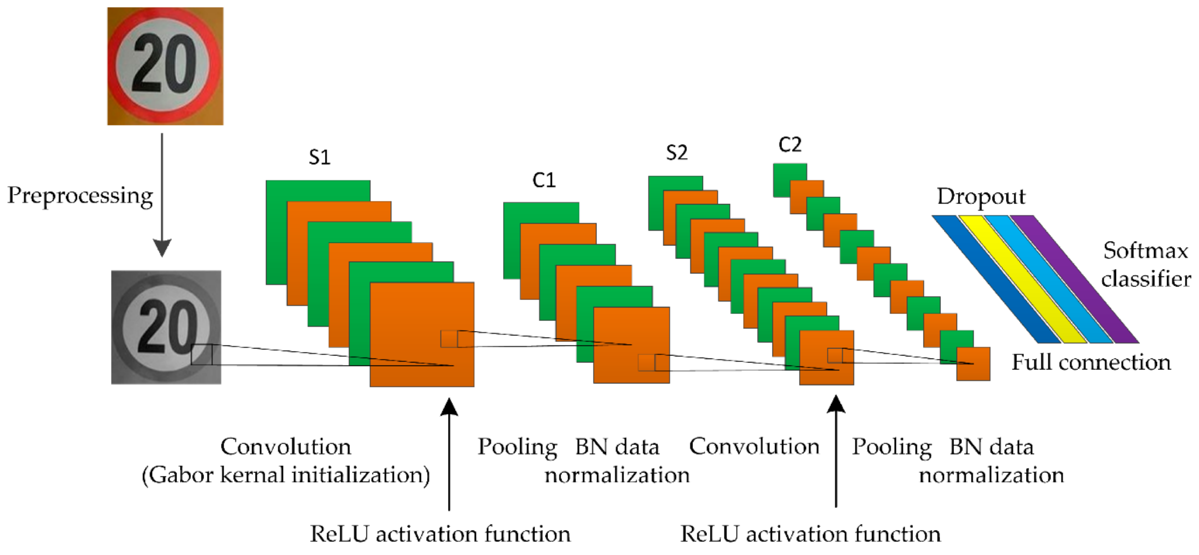
Introduction:

A CNN is a class of neural networks, defined as multi-layered neural networks designed to detect complex features in data. It is a feed-forward network. During the training process, the network will process the input through all the layers, compute the loss to understand how far the predicted label of the image is falling from the correct one, and propagate the gradients back into the network to update the weights of the layers.

By iterating over a huge dataset of inputs, the network will "learn" to set its weights to achieve the best results. A forward function computes the value of the loss function, and the backward function computes the gradients of the learnable parameters.

To train our model, we loop over our data iterator, feed the input images to the network, and optimize. Then we can test the model with batch of images from the test set we have created.

A CNN basically consists of the input layer, hidden layers, and output layer.

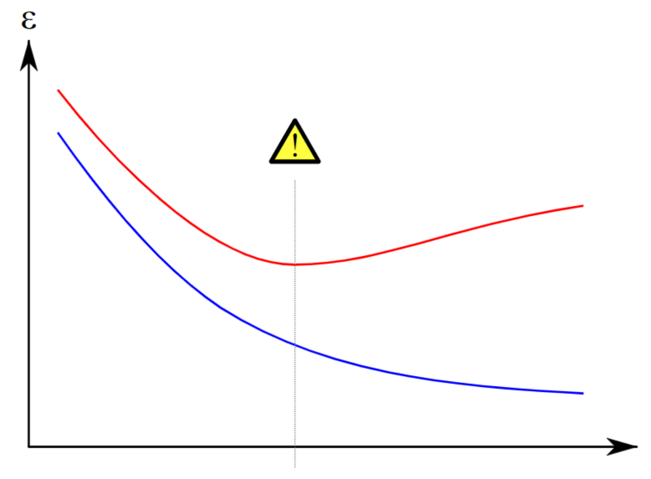


Important terms in CNNs-

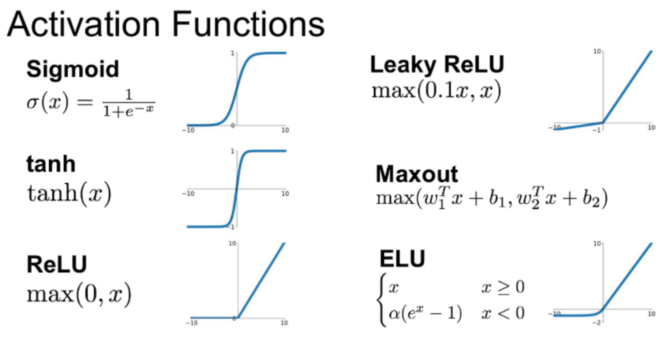
1. ***Loss function:*** It computes a value that estimates how far away the output is from the target. The main objective is to reduce the loss function's value by changing the weight vector values through backpropagation in neural networks. Loss function gives us the understanding of how well a model behaves after each iteration of optimization on the training set. The accuracy of the model is calculated on the test data and shows the percentage of the right prediction.
2. ***Optimizer:*** The optimizers have some elements of the gradient descent. By changing the model parameters, like weights, and adding bias, the model can be optimized.
3. ***Learning Rate:*** Learning rate (lr) sets the control of how much you are adjusting the weights of our network with respect the loss gradient. Usually, it is set as 0.001. The

lower it is, the slower the training will be. The learning rate will decide how big the steps should be to change the parameters.

4. ***Weights:*** The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights. The vector of weights and the bias are called **filters** and represent particular features of the input (e.g., a particular shape).
5. ***Bias:*** the constant which is added to the product of features and weights is called bias. It is used to offset the result. It helps the models to shift the activation function towards the positive or negative side.
6. ***Epoch:*** Epoch can be understood as the number of times the algorithm scans the entire data. For example, if we set epoch = 10 then the algorithm will scan the entire data ten times.
7. ***Backpropagation:*** It is an algorithm widely used in the training of feedforward neural networks for supervised learning. Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network for a single input-output example. This makes it feasible to use gradient methods for training multi-layer networks, updating weights to minimize loss; commonly one uses gradient descent or variants such as stochastic gradient descent. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, iterating backwards one layer at a time from the last layer to avoid redundant calculations of intermediate terms in the chain rule.
8. ***Overfitting:*** An overfitted model is a statistical model that contains more parameters than can be justified by the data. The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e., the noise) as if that variation represented underlying model structure. In the image below: *training error* is shown in blue, *validation error* in red, both as a function of the number of training epochs. If the validation error increases (positive slope) while the training error steadily decreases (negative slope), then a situation of overfitting may have occurred. The best predictive and fitted model would be where the validation error has its global minimum.



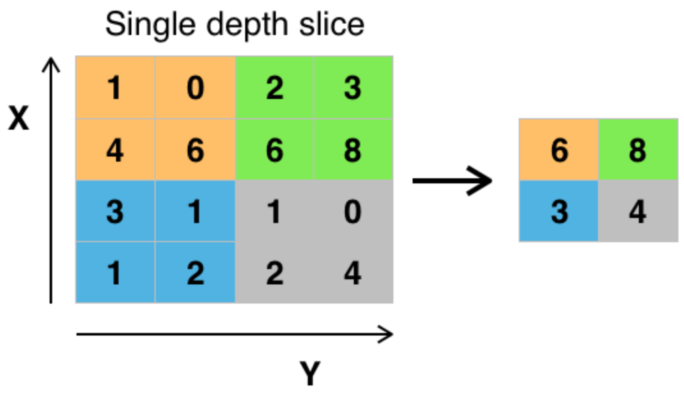
9. ***Activation function:*** In a multilayer neural network, there is a functional relationship between the output of the upper node and the input of the lower node. This function is called the activation function.



<u>Layers in CNNs-</u>

A **convolutional neural network (CNN)** consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of **convolutional layers** that convolve with a multiplication or other dot product. The activation function is commonly a *REctified Linear Unit (RELU) layer* and is subsequently followed by additional convolutions such as **pooling layers**, **fully connected layers** and **normalization layers**, referred to as *hidden layers* because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product.

1. ***<u>Input Layer:</u>*** When programming a CNN, the input is a tensor with shape (number of images, (image width, image height), image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images, (feature map width, feature map height), feature map channels).
2. ***<u>Convolution Layer</u>*****:** Convolutional layers convolve the input and pass its result to the next layer. The convolution operation brings a solution to the problem arising from the presence of a huge number of input data (i.e., the number of pixels) as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.
3. ***<u>Pooling:</u>*** Pooling layers *reduce the dimensions of the data* by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. *Max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. *Average pooling* uses the average value from each of a cluster of neurons at the prior layer. This is a Max pooling.



4. ***Fully connected Layers:*** These connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.
5. ***Normalization Layer:*** Normalizing a set of data transforms the set of data to be on a similar scale. Normalization can help training of our neural networks as the different features are on a similar scale, which helps to stabilize the gradient descent step, allowing us to use larger learning rates or help models converge faster for a given learning rate. The different types are- Batch Normalization, Weight Normalization, Layer Normalization, Group Normalization, Weight Standardization.
6. ***Output Layer:*** The output layer is the final layer in the neural network where desired predictions are obtained.

Traffic Sign Classification CNN

The following CNN will classify 4 different traffic sign images of priority road, no entry, stop, and men at work signs from the **German Traffic Sign Recognition Benchmark (GTSRB)** dataset which contains more than 50,000 images of more than 40 traffic signs.

Checking for CUDA

Whenever you initialize the batch of images, it is on the CPU for computation by default. The below code will check whether a GPU is present. If CUDA is present, it will route the tensor to the GPU for computation. Since we have only a CPU in our system, we use Google Colab, which provides free GPU.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cpu')
```

Splitting the Dataset

How well the model can learn depends on the variety and volume of the data. We need to divide our data into a training set and a validation set. We take the file indices of the 4 traffic signs and split them into separate folders named train, val, and test respectively.

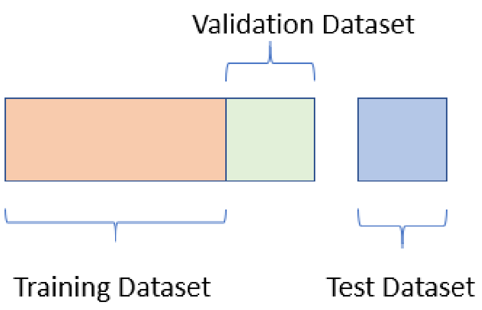
Training dataset: The model learns from this dataset's examples. It fits a parameter to a classifier.

Validation dataset: The examples in the validation dataset are used to tune the hyperparameters, such as learning rate and epochs. The aim of creating a validation set is to avoid large overfitting of the model. It is a checkpoint to know if the model is fitted well with the training dataset.

Test dataset: This dataset test the final evolution of the model, measuring how well it has learned and predicted the desired output. It contains unseen data.

```
[6]  class_names = ['men at work', 'priority road', 'stop', 'no entry']

     class_indices = [25, 12, 14, 17]
```

```
[7]  !rm -rf data

     DATA_DIR = Path('data')

     DATASETS = ['train', 'val', 'test']

     for ds in DATASETS:
       for cls in class_names:
         (DATA_DIR / ds / cls).mkdir(parents=True, exist_ok=True)
```



The more training data we have, so more accurate the output is. The number of images we have for each dataset is-

```
men at work: 1500
no entry: 2100
priority road: 780
stop: 1110
```

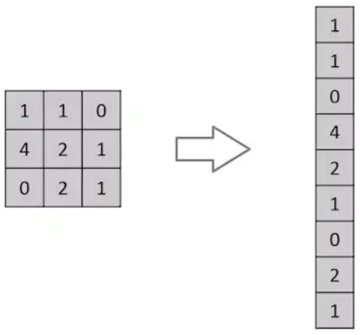
After splitting the datasets into the 3 folders

```
[ ]  dataset_sizes = {d: len(image_datasets[d]) for d in DATASETS}
     class_names = image_datasets['train'].classes

     dataset_sizes

     {'train': 6109, 'val': 1050, 'test': 1048}
```

It shows the number of images we have for training, validation, and testing respectively.

Image Pre-processing

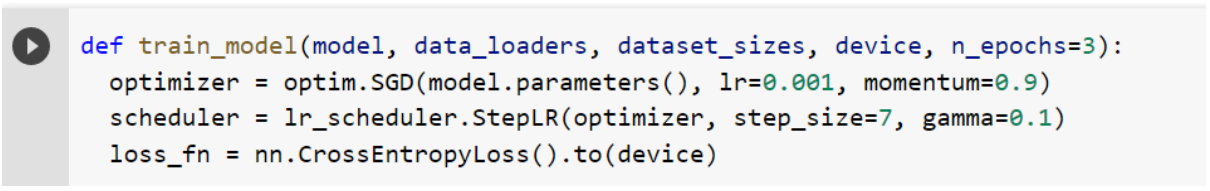
Images in a dataset do not usually have the same pixel intensity and dimensions. So we will pre-process the dataset by standardizing the pixel values. The next required process is transforming raw images into tensors so that the algorithm can process them. We then normalize the dataset to make the model training stable and fast.



Creating CNN model

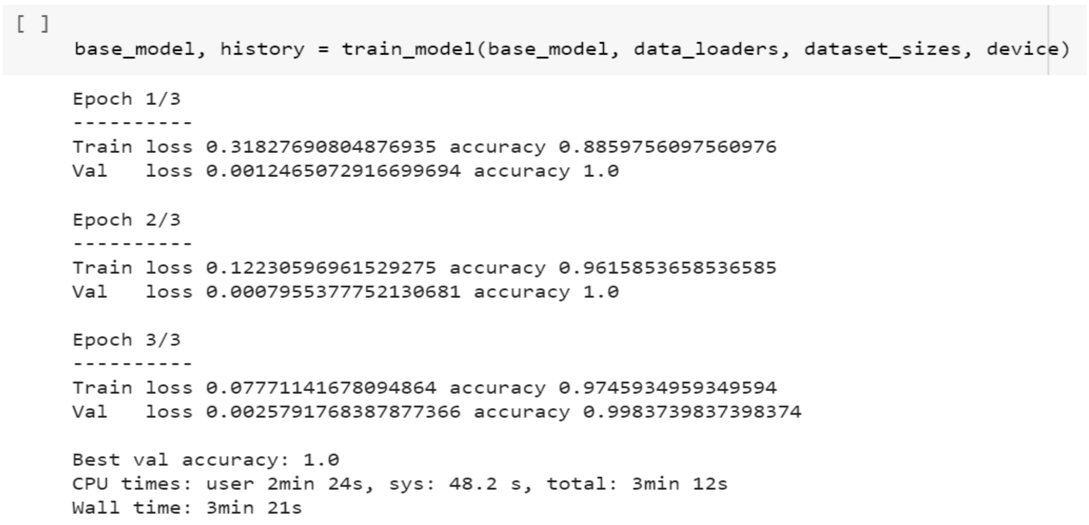
We then create the traffic sign classification model and train the data with parameters.

Learning rate=0.001, momentum=0.9, step size=7, gamma=0.1, and loss function as CrossEntropyLoss() function.



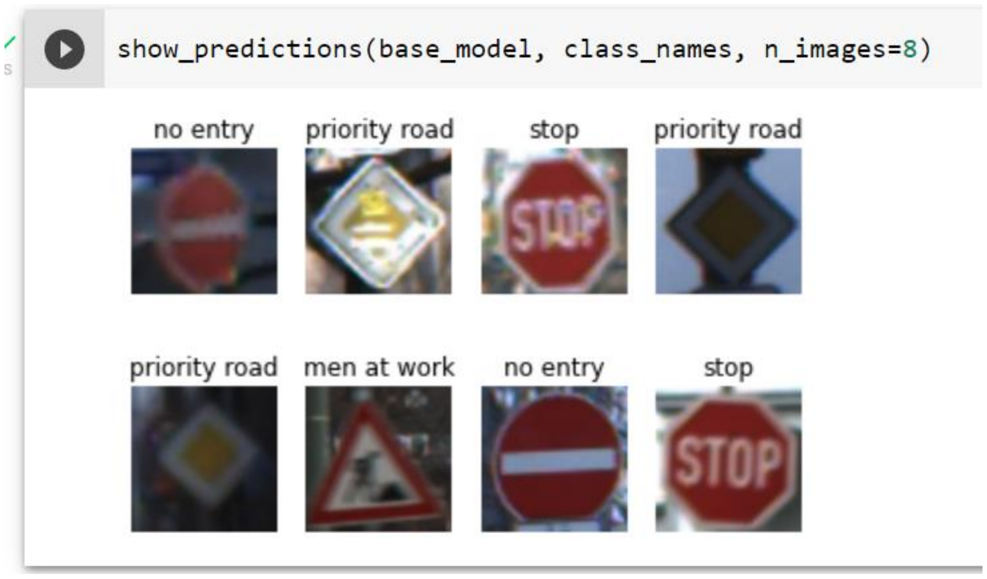
```
def train_model(model, data_loaders, dataset_sizes, device, n_epochs=3):
  optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
  scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
  loss_fn = nn.CrossEntropyLoss().to(device)
```

After running the epochs, we have the loss in data as well as accuracy-



```
[ ]
    base_model, history = train_model(base_model, data_loaders, dataset_sizes, device)

    Epoch 1/3
    ----------
    Train loss 0.31827690804876935 accuracy 0.8859756097560976
    Val   loss 0.0012465072916699694 accuracy 1.0

    Epoch 2/3
    ----------
    Train loss 0.12230596961529275 accuracy 0.9615853658536585
    Val   loss 0.0007955377752130681 accuracy 1.0

    Epoch 3/3
    ----------
    Train loss 0.07771141678094864 accuracy 0.9745934959349594
    Val   loss 0.0025791768387877366 accuracy 0.9983739837398374

    Best val accuracy: 1.0
    CPU times: user 2min 24s, sys: 48.2 s, total: 3min 12s
    Wall time: 3min 21s
```

Finally, we choose 8 random images from the test dataset and predict their classification-



```
show_predictions(base_model, class_names, n_images=8)
```

Uses

Traffic Sign Classification is very useful in Automatic Driver Assistance Systems. A convolutional neural network is used to examine and check visual imagery, and so it is used to train the image classification and recognition model because of its high accuracy and precision.

Conclusion

The above model is simple and does the classification quite accurately on the GTSRB dataset, and the model can successfully predict traffic signs accurately even if the background of the image is not much clear. Convolutional Neural Network (CNN) is used to train the model. The final accuracy on the test dataset is around 96%. The benefits of "Traffic Sign classification and detection system" are generally focused on driver convenience. Despite the advantages of traffic sign classification, there are drawbacks. There can be times when the traffic signs are covered or not visible clearly. This can be dangerous as the driver won't be able to keep a check on his vehicle speed and can lead to accidents, endangering other motorists or pedestrians, demanding further research.

References

https://colab.research.google.com/drive/1Lk5R4pECDxDhd1uXcv26YRQ02fb_mrL9?usp=sharing#scrollTo=Jjc6AU2SScpI

https://learn.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-train-model

https://www.kaggle.com/code/androbomb/using-cnn-to-classify-images-w-pytorch

https://www.pluralsight.com/guides/image-classification-with-pytorch

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53