Machine Learning and deep learning learned representation from the data like background texture, nose etc.

Machine Learning

“searching for useful representations of some input data, within a predefined space of possibilities(hypothesis space), using guidance from a feedback signal.”

Keep in mind that machine learning can only be used to memorize patterns that are present in your training data. You can only recognize what you’ve seen before. Using machine learning trained on past data to predict the future is making the assumption that the future will behave like the past. That often isn’t the case.

Hypothetical space->Set of predefined operation or possible hypothesis

Shallow learning-> 1 or 2 layers

Deep Network-> multistage information distillation process where input information purified from successive filters(layers)

Weights->transformation implemented by a layer is *parameterized* by its weights (Weights are also sometimes called the *parameters* of a layer.)

Initially deep learning prediction is bad or loss is high as weights were assigned random values. Loss function takes prediction and true value, computes loss and do back propagation using optimizer to decrease the loss and update weights accordingly. This all done during training.

Feature engineering in machine learning->manually engineer good layers of representations for their data. But this is done automatically by deep learning that’s why deep learning is better than the machine learning

These are the two essential characteristics of how deep learning learns from data:

the *incremental, layer-by-layer way in which increasingly complex representations are developed*,

and the fact that *these intermediate incremental representations are learned jointly*

To make the network ready for training, we need to pick three more things, as part

of the *compilation* step:

 *A loss function*—How the network will be able to measure its performance on

the training data, and thus how it will be able to steer itself in the right direction.

 *An optimizer*—The mechanism through which the network will update itself based on the data it sees and its loss function.

 *Metrics to monitor during training and testing*—Here, we’ll only care about accuracy

(the fraction of the images that were correctly classified).

3D shape explanation:

If all data at once without batch then, then 0 axis is all data or sample axis

(28,28)

(sample, dim, dim) -> (60000, 28, 28)

If perform batching then 0 axis is batch axis

(batchsize dim, dim) -> (128, 28, 28) (if one batch contain 128 data points or sample points)

(128, 28\*28) -> (128,784)

Batch formula -> batch = train\_images[128 \* n:128 \* (n + 1)] -> Illustration given below

for first batch n=0

128\*0:128\*(0+1)

0:128

Backpropagation:

Backpropagation starts with the final loss value and works backward from the top layers to the bottomlayers, applying the chain rule to compute the contribution that each parameterhad in the loss value.

WHOLE NEURAL NETWORK PROCESS:

the network, composed of layers that are chained together, maps the input data to predictions. The loss function

then compares these predictions to the targets, producing a loss value: a measure of how well the network’s predictions match what was expected. The optimizer uses this loss value to update the network’s weights.

About different tensors and layers used as per those tensors:

For instance, simple vector data, stored in 2D tensors of shape (samples, features), is often processed by *densely connected* layers, also called *fully connected* or *dense* layers (the Dense class in Keras). Sequence data, stored in 3D tensors of shape (samples, timesteps, features), is typically processed by *recurrent* layers such as an LSTM layer. Image data, stored in 4D tensors, is usually processed by 2D convolution layers (Conv2D).

SELECTION OF LOSS FUNCTION:

Binary crossentropy for a two-class classification

Categorical crossentropy for a many-class classification problem,

Mean squared error (mse) for a regression problem,

Connectionist temporal classification (CTC) for a sequence-learning problem

SELECTION OF ACTIVATION FUNCTION:

If more than two class classification and categorized the data -> Softmax

If between two classes and find probability-> sigmoid(score between 0 and 1)

If regression than-> no last layer activation, only relu in hidden layers

WHY ACTIVATION FUNCTION IS REQUIRED:

If no activation function, then layers will learn linear transformation of the input data(less hypothesis space).

In order to get access to a much richer hypothesis space that would benefit from

deep representations, you need a non-linearity which we get using activation function or activation function. relu is the most popular activation function in deep learning

WORKFLOW IN KERAS:

***Developing with Keras: a quick overview***

1 Define your training data: input tensors and target tensors.

2 Define a network of layers (or *model* ) that maps your inputs to your targets.

3 Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.

4 Iterate on your training data by calling the fit() method of your model.

NORMALIZATION:

Data centered to mean 0 and std 1

If different ranges-> normalize the data

When features in the input data have values in different ranges, each feature should be scaled independently(normalized) as a preprocessing step.

Otherwise vectorise the data

If softmax->one hot encode labels

WHY VALIDATION DATA IS REQUIRED?

The reason is that developing a model always involves tuning its configuration: for

example, choosing the number of layers or the size of the layers (called the *hyperparameters*

of the model, to distinguish them from the *parameters*, which are the network’s

weights). You do this tuning by using as a feedback signal the performance of

the model on the validation data. In essence, this tuning is a form of *learning*: a search

for a good configuration in some parameter space. As a result, tuning the configuration

of the model based on its performance on the validation set can quickly result in

*overfitting to the validation set*, even though your model is never directly trained on it.

Central to this phenomenon is the notion of *information leaks*. Every time you tune

a hyperparameter of your model based on the model’s performance on the validation

set, some information about the validation data leaks into the model. If you do this

only once, for one parameter, then very few bits of information will leak, and your validation

set will remain reliable to evaluate the model. But if you repeat this many

times—running one experiment, evaluating on the validation set, and modifying your

model as a result—then you’ll leak an increasingly significant amount of information

about the validation set into the model.

At the end of the day, you’ll end up with a model that performs artificially well on

the validation data, because that’s what you optimized it for. You care about performance

on completely new data, not the validation data, so you need to use a completely

different, never-before-seen dataset to evaluate the model: the test dataset. Your

model shouldn’t have had access to *any* information about the test set, even indirectly.

Conclusion:

If we want to test our mode, training data is obviously useless as model will be specific to that data. To do hyperparameter tuning for better results, we do this on validation data. But if we do hyperparameter tuning many times for best results, at the end our model also become somewhat specific to validation data. So at the end for final results or verification or testing, test data is required.

K-Fold Validation:

If data samples are smaller in size like bouston housing dataset, then it is difficult to split training data into training and validation data. To do so, we use k fold validation where it splits available training data into k portions of equal size typically *K* = 4 or 5), instantiating *K* identical models, and training each one on *K* – 1 partitions while evaluating on the remaining partition. The validation score for the model used is then the average of the *K* validation scores obtained. In terms of code, this is straightforward.

Types of Validation (Summarized):

 *Maintaining a hold-out validation set*—The way to go when you have plenty of

data

 *Doing K-fold cross-validation*—The right choice when you have too few samples

for hold-out validation to be reliable

 *Doing iterated K-fold validation*—For performing highly accurate model evaluation

when little data is available

Just pick one of these. In most cases, the first will work well enough.

4 Categories of Machine Learning:

Supervised learning is just the tip of the iceberg—machine learning is a vast field with a complex subfield taxonomy. Machine-learning algorithms generally fall into four broad categories.

***Supervised learning***

This is by far the most common case. It consists of learning to map input data to

known targets (also called *annotations*), given a set of examples (often annotated by

humans).

***Unsupervised learning***

This branch of machine learning consists of finding interesting transformations of the input data without the help of any targets, for the purposes of data visualization, data compression, or data denoising, or to better understand the correlations present in the data at hand

***Self-supervised learning***

Self-supervised learning is supervised learning without human-annotated labels—you can think of it as supervised learning without any humans in the loop. There are still labels involved (because the learning has to be

supervised by something), but they’re generated from the input data, typically using a heuristic algorithm.

***Reinforcement learning***

In reinforcement learning,

an *agent* receives information about its environment and learns to choose actions that

will maximize some reward. For instance, a neural network that “looks” at a videogame

screen and outputs game actions in order to maximize its score can be trained

via reinforcement learning. In time, however, we expect to see reinforcement learning take over an increasingly large range of real-world applications: self-driving cars, robotics, resource management, education, and so on. It’s an idea whose time has come, or will come soon.

VECTORIZATION:

Data is to be vectorized or one hot encoded (0 and 1 in either integer or float32) before feeding to neural Network. Means the index where data is present is replaced with 1 and other with 0.

It is mostly used for categorical data or for binary and multiclass classification

NORMALIZATION:

If data is available in different ranges, then to equally spaced data from some center value, we do normalization. Data is centered around mean 0 and std 1. Purpose is:

*Take small values*—Typically, most values should be in the 0–1 range.

*Be homogenous*—That is, all features should take values in roughly the same range.

FEATURE ENGINEERING:

The data needs to be presented to the model in a way that will make the model’s job easier. Before deep learning, it was used in ML, where we apply feature engineering on raw data to make it look better and model can perform better on it. After Deeplearning, deeplearning models are much more efficient as their no obligatory of feature engineering. But if it is done, it will help in achieving better results.

REGULARIZATION (HOW TO PREVENT OVERFITTING):

The processing of fighting overfitting this way is called *regularization.*

1. ***Reducing the network’s size***

The simplest way to prevent overfitting is to reduce the size of the model: the number of learnable parameters in the model (which is determined by the number of layers and the number of units per layer).

2. ***Adding dropout***

*Dropout* is one of the most effective and most commonly used regularization techniques for neural networks

At training time, we zero out at random a fraction of the values(units) in the matrix(layers):

At test time, no units are dropped out; instead, the layer’s output values are scaled down by a factor

equal to the dropout rate, to balance for the fact that more units are active than at training time.

To recap, these are the most common ways to prevent overfitting in neural networks:

 Get more training data.

 Reduce the capacity of the network.

 Add weight regularization.

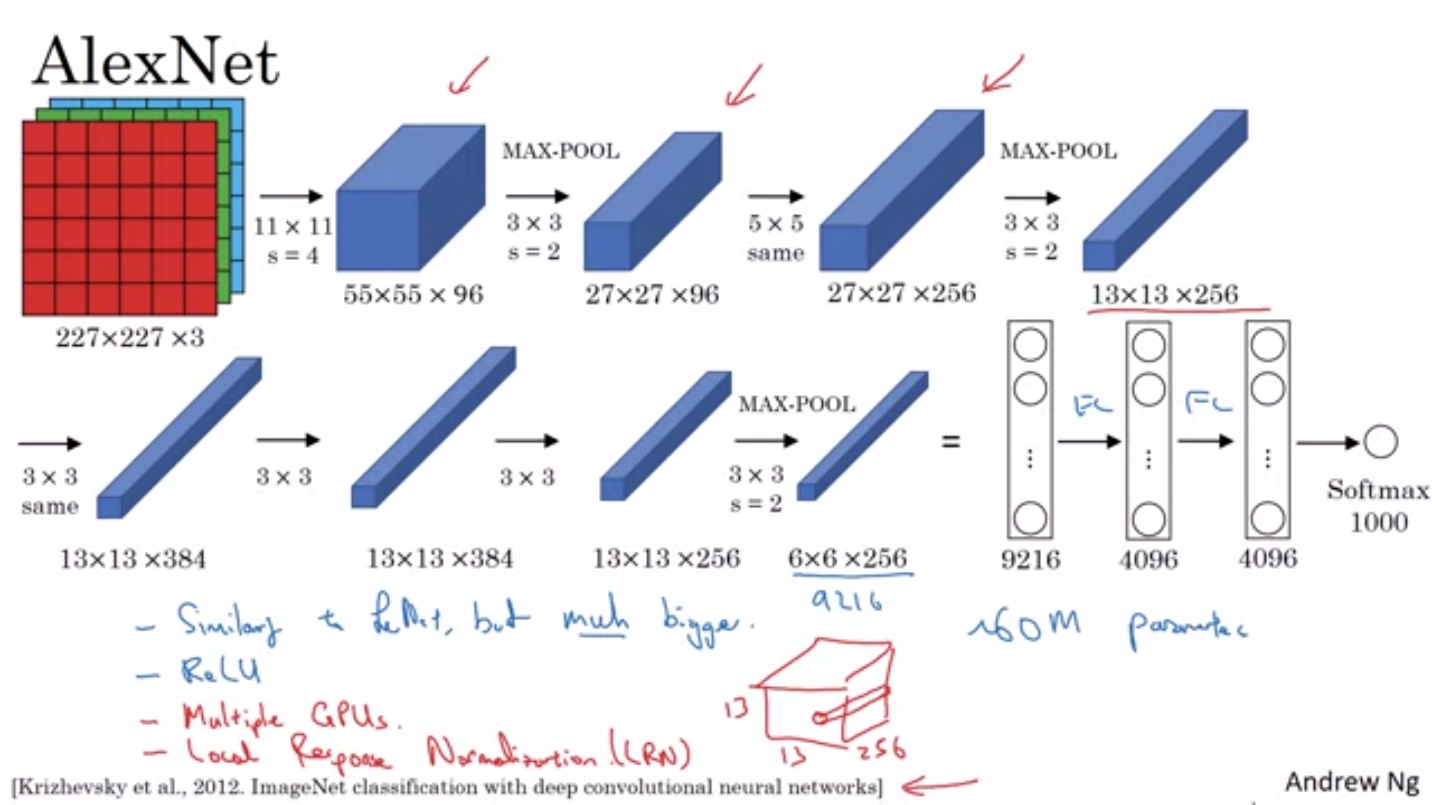
 Add dropout.

Normally convolution output contains no of filters

And when pad, dimensionality same and output contains no of filters as applied

P=(f-1)/2

Pooling layers contains same no of channels in the output as of input but reduce dimensionality



Here when applied simple convolution of fitter size 11x11 and of s=4 -> Dimensional reduction and no of filters as per given which is not shown here, (It would be like 96 of k= 11x11x3)

After that when applied maxpooling of k=3x3x96 and s=2->dimensional reduction to avoid overfitting but same no of channels as of input

After that when applied padding to make output same as of input of 256 no of k=5x5x96->27x27x256

P=(f-1)/2 -> P=(5-1)2 = 2

And 256 in output because in simple convolution and padding, output 3rd axis (nc) contains no of filters while pooling contains channel in nc

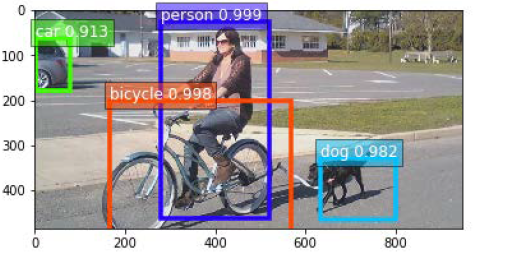
Computer Vision:

**Image classification** assigns one or more labels to an image.

**Object detection:** detects and classifies objects in an image. the **object detection** algorithm predicts both the class and the location of each object in the image. Of course, the algorithm can only detect object

classes present in the training dataset. The location of each object is defined by a set of

four coordinates, called a **bounding box**



**Semantic segmentation** assigns every pixel of an image to a specific class. This is a much harder problem than image classification (which only considers the full image) or object detection (which only focuses on specific parts of the image).



In short,

• Class labels for **image classification**

• Bounding boxes and class labels for **object detection**

• Segmentation masks and class labels for **semantic segmentation**

**Transfer Learning:**

**Transfer learning** is a machine **learning** method where a model developed for a task is reused as the starting point for a model on a second task.

Object classification:

Separated object into classes like people and car

Object detection:

Give classes labels like person1, person 2 etc. Object detection, also called **object localization**,

Instance segmentation:

Beyond step of localization. Mask image with labels

Image localization:

Location finding of object using bounding box

Pose estimation:

Check orientation or pose of image

Instance tracking:

Checking where object moving in video in frames

Action Recognition:

Recognizing emotion of object (swimming, dancing etc) from a predefined set of data

Motion estimation:

Recognizing moving element of object

Content aware image edition:

Making your image very good like making it is enhancing, light good from environment, removing blurriness

Scale Invariant feature transformation:

Reconstruct image from different features which was extracted while extracting features from object. This is done by pixel to pixel comparision. For this, machine learning is used

MSE is the avg version of L2 loss and MAE is the avg version of L1 loss

Latent space: encoder, image se kaam ke features nikaal ker baaki chhordein ya text se meaning full conversion extract kerlein khaali, ya compress hojae object feature nikaalne ke baad. This results in feature vectors, means encoders results in feature vectors

Generative task: abstract image se detail image bnana

The number of true positives: True positives (TP) determine how many

predictions match with a ground truth box of the same class.

The number of false positives: False positives (FP) determine how many

predictions do not match with a ground truth box of the same class.

The number of false negatives: False negatives (FN) determine how many

ground truths do not have a matching prediction.

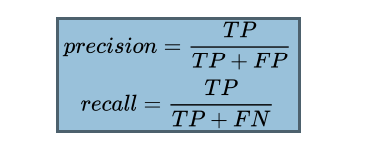
**Bounding box precision**: Providing the correct bounding box (not too large or

too narrow)

**Recall**: Finding all the objects (not missing any objects)

**Class precision**: Outputting the correct class for each object (not mistaking a cat

for a dog)



The core idea of YOLO is this: **reframing object detection as a single regression problem**

For each part of the grid, we will define *B* bounding boxes. Then, our only task will be to

predict the following for each bounding box:

* The center of the box
* The width and height of the box
* The probability that this box contains an object
* The class of said object

Since all those predictions are numbers, we have therefore transformed the object detection

problem into a regression problem

**Intersection over Union** (**IoU**): which measures how well two sets overlap (in our case, the sets of pixels represented by the boxes) also called as **Jaccard index**,

Anchor boxes:

Anchor boxes (also called **priors**) are a set of bounding box sizes that are decided upon

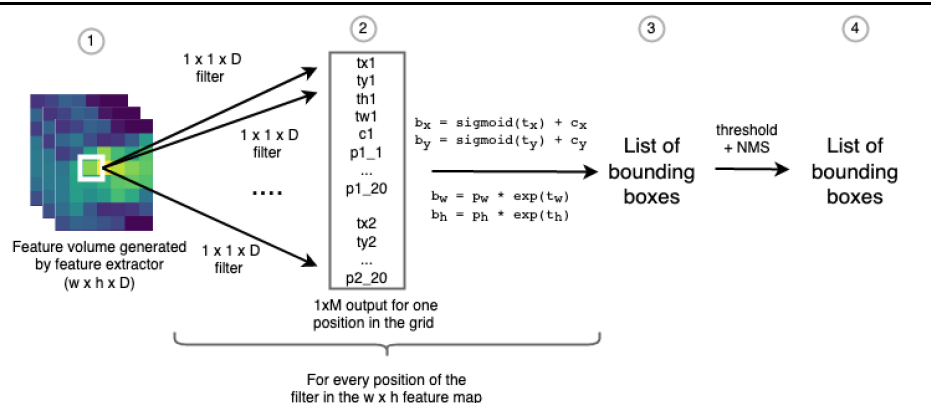
before training the network. For instance, when training a neural network to detect

pedestrians, tall and narrow anchor boxes would be picked

Yolo pathway:

Image -> CNN backbone -> feature volume -> CNN layer -> anchor box corrections, objectness scores, and class probabilities -> Using this output, compute the coordinates of the bounding boxes

->Filter out the boxes with a low threshold, and post-process the remaining ones using NMS.



Faster R-CNN architecture:

