

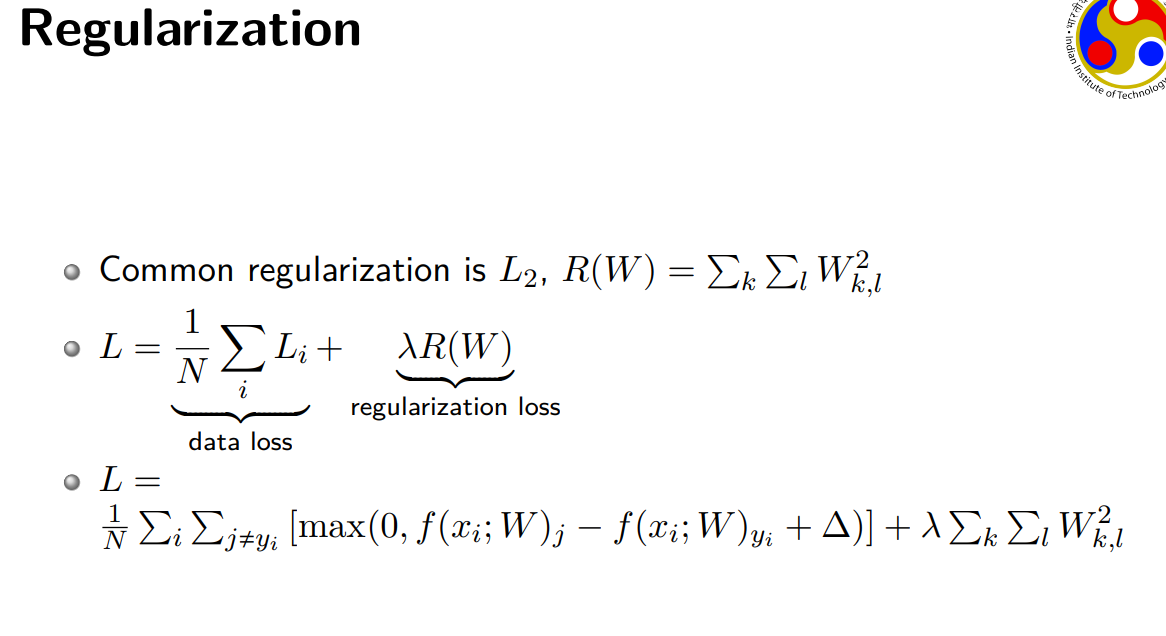
**Multiclass Support Vector Machine loss**

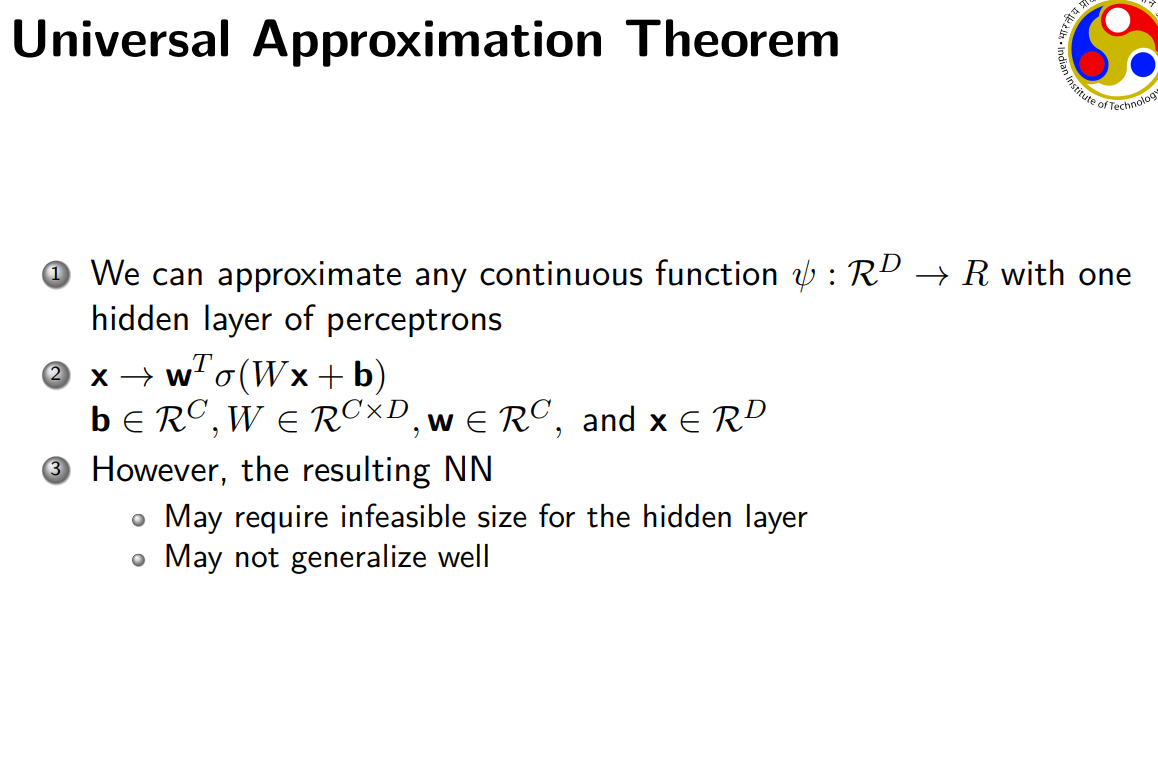
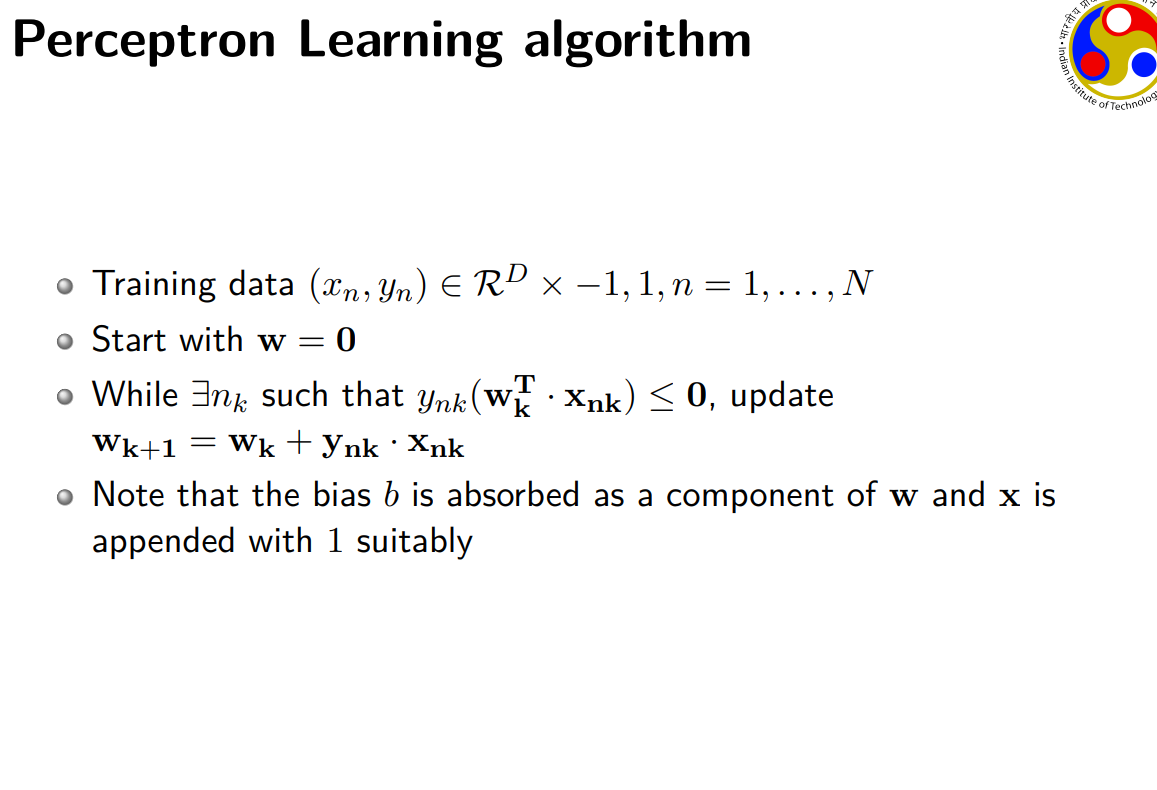
There are several ways to define the details of the loss function. As a first example we will first develop a commonly used loss called the Multiclass Support Vector Machine (SVM) loss. The SVM loss is set up so that the SVM “wants” the correct class for each image to a have a score higher than the incorrect classes by some fixed margin Δ. Notice that it’s sometimes helpful to anthropomorphise the loss functions as we did above: The SVM “wants” a certain outcome in the sense that the outcome would yield a lower loss (which is good).

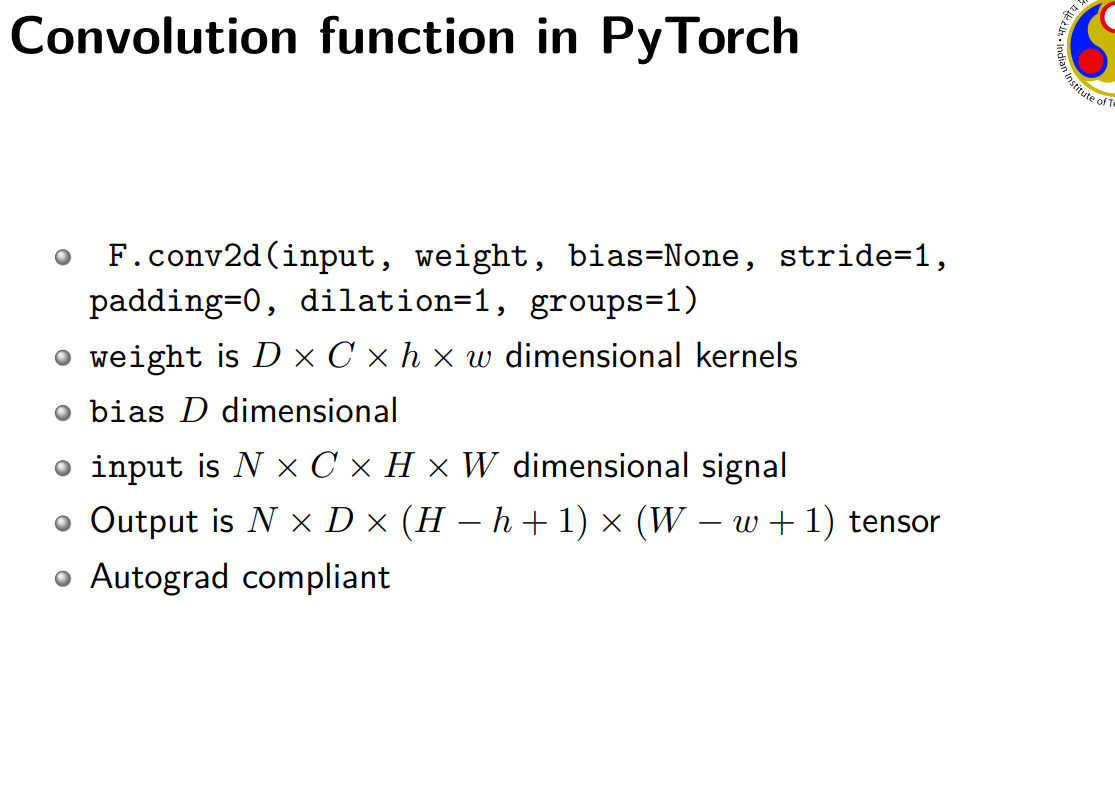
Li=∑j≠yimax(0,sj−syi+Δ)

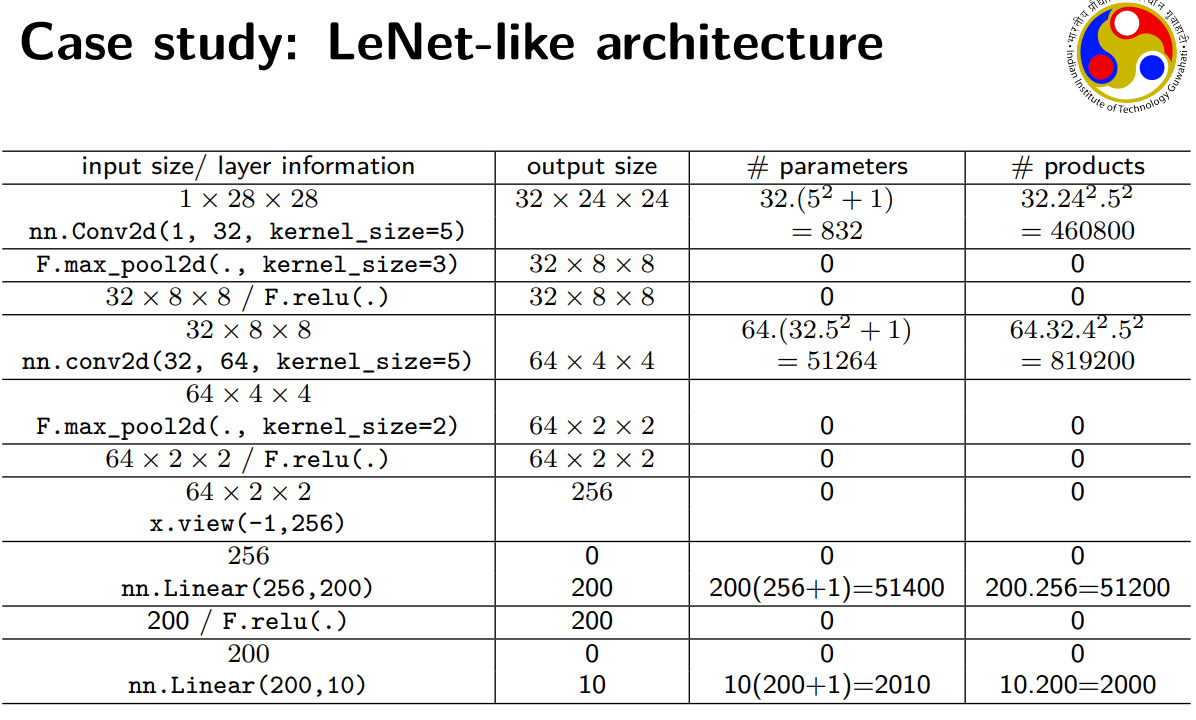
**Example.** Let’s unpack this with an example to see how it works. Suppose that we have three classes that receive the scores s=[13,−7,11], and that the first class is the true class (i.e. yi=0). Also assume that Δ (a hyperparameter we will go into more detail about soon) is 10. The expression above sums over all incorrect classes (j≠yij≠yi), so we get two terms:

Li=max(0,−7−13+10)+max(0,11−13+10)









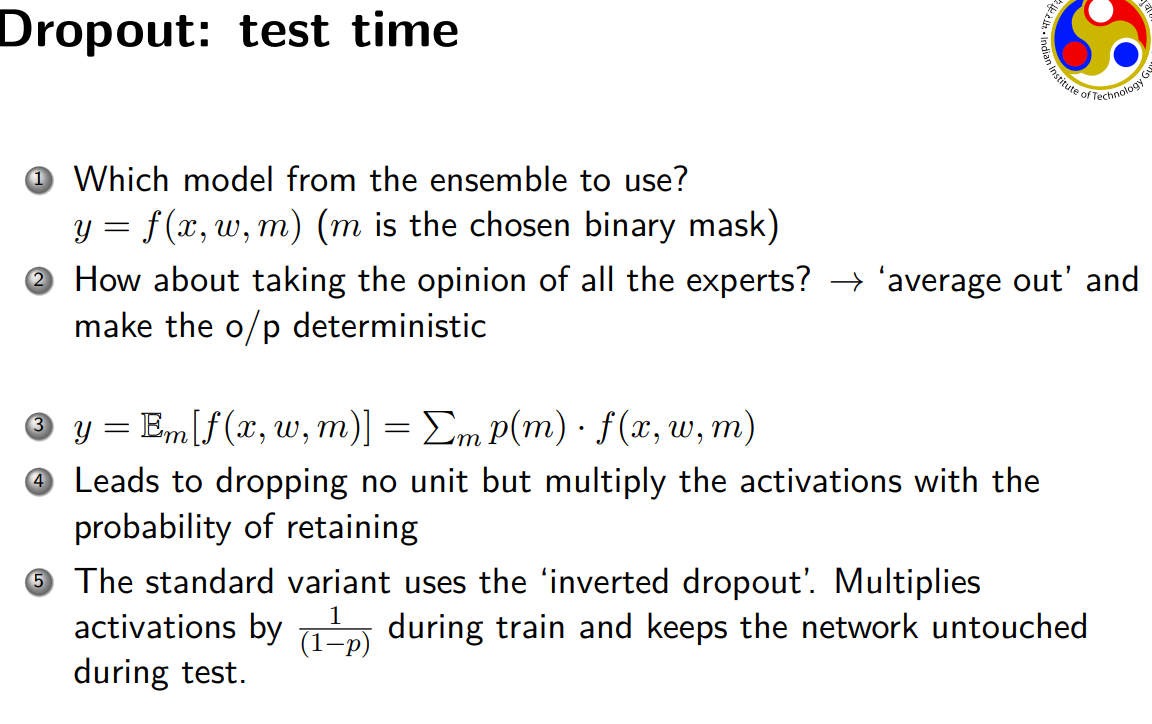
**Auxiliary Classifiers** are type of architectural component that seek to improve the convergence of very deep networks. They are classifier heads we attach to layers before the end of the network. The motivation is to push useful gradients to the lower layers to make them immediately useful and improve the convergence during training by combatting the vanishing gradient problem. They are notably used in the Inception family of convolutional neural networks.

**Summary**

1. A residual network is formed by stacking several residual blocks together.
2. The residual blocks create an identity mapping to activations earlier in the network to thwart the performance degradation problem associated with deep neural architectures.
3. The skip connections help to address the problem of vanishing and exploding gradients.

In fact, any constant initialization scheme will perform very poorly. Consider a *neural network* with two hidden units, and assume we initialize all the biases to 0 and the weights with some constant *α*. If we forward propagate an input (*x*1​,*x*2​) in this network, the output of both hidden units will be *relu*(*αx*1​+*αx*2​). Thus, both hidden units will have identical influence on the cost, which will lead to identical gradients. Thus, both neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things.

[Initializing neural networks - deeplearning.ai](https://www.deeplearning.ai/ai-notes/initialization/index.html)



[Parameter optimization in neural networks - deeplearning.ai](https://www.deeplearning.ai/ai-notes/optimization/index.html)