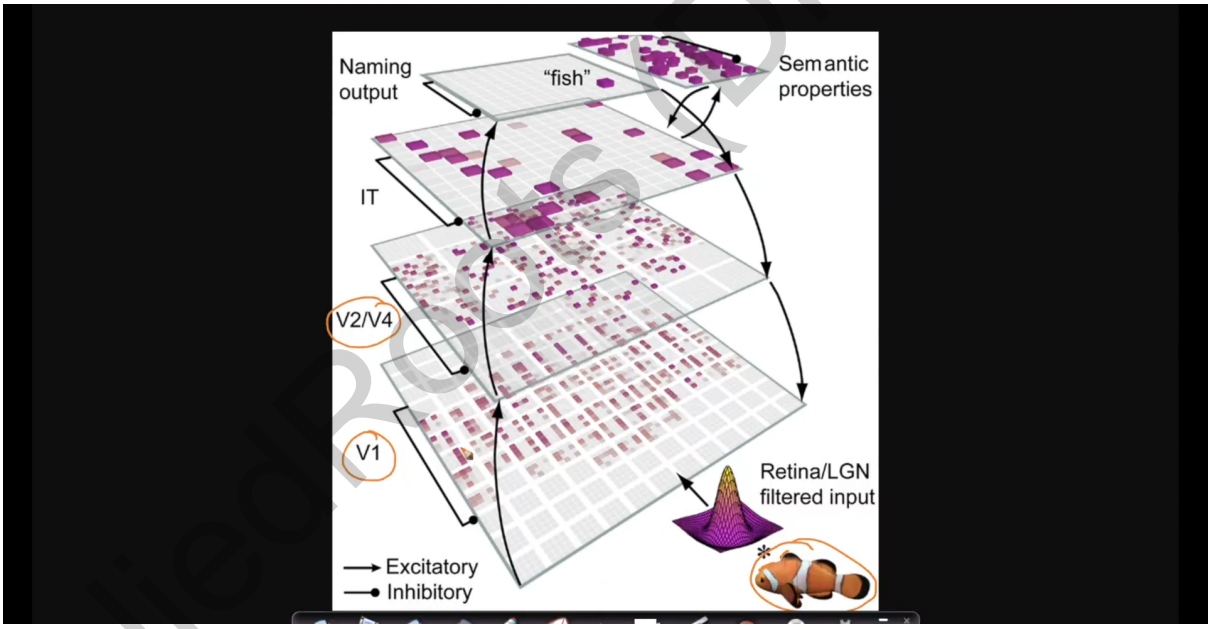
60.1 Biological inspiration: Visual Cortex

ConvNets are designed for visual tasks, like Object Recognition, Object Detection,etc. Computer vision can be thought of as the intersection of Visual tasks and AI. Some of the tasks in computer vision can be done by computers as good as humans due to deep learning.

Key findings from Hubel and Wiesel Studies:

- Some neurons in the visual cortex fire when presented with light in different angles.
- There are edge, color, motion, depth detecting neurons.
- Hierarchical structure is present among the layers in the visual cortex.

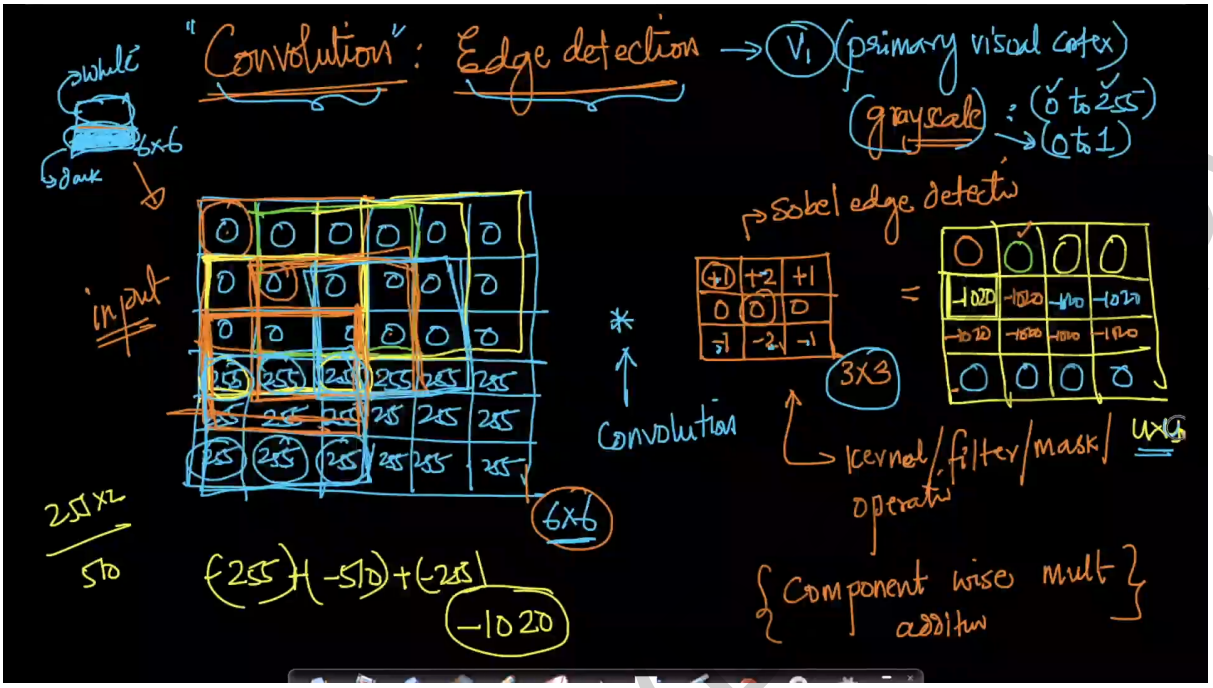
This shows the hierarchy in the visual cortex for Classification of Fish



Notice that V1 layer is connected to V2/V4 and so on, this is the hierarchy we are talking about. At the top of the hierarchy we are predicting fish. Most of the research in the field of Image Processing and the convolutional layers which just follow this are based on these key findings from the Hubel and Wiesel studies as well.

60.2 Convolution:Edge Detection on images

Finding edges in an image using Sobel Horizontal Edge Detector.

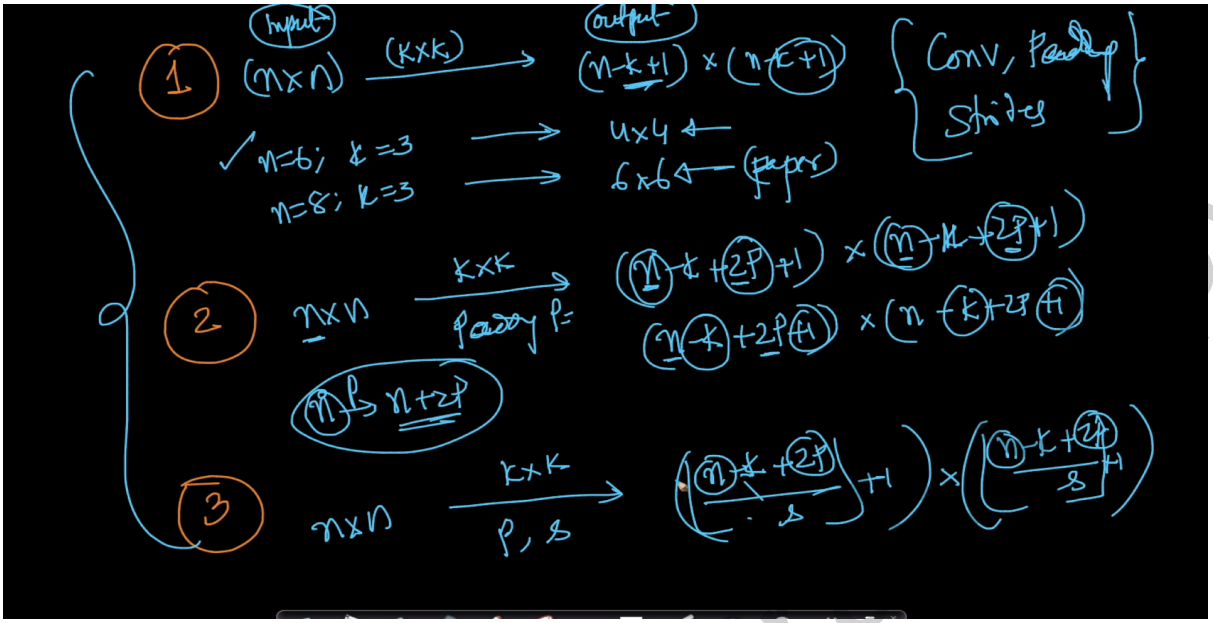


Convolution can be thought of as a generalization of dot product. The output of a convolution is scaled such that max value and min value in the output will be 255,0 respectively.Notice that the size of the output in the above figure is 4x4 while the kernel is of 3x3 size and input is 6x6 size.

The kernel slides over the input image by one pixel and does the dot product(component wise multiplication and addition) between kernel and input pixels. It will help to try using the input in the above diagram and figure out the values on the right side output.

This image is the output when the sobel operator is used to find the edges. We can observe that the edges are clearly highlighted, and here Sobel Vertical edge detector is used alongside the Sobel Horizontal edge detector.

60.3 Convolution:Padding and strides



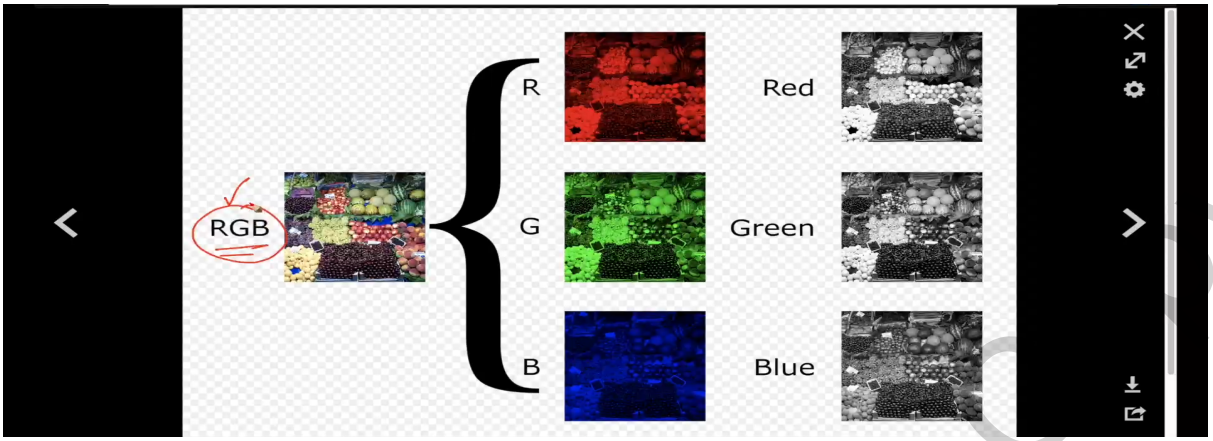
Here we will be talking about the output sizes when a convolution operation is done. Above are the formulas which can tell this and below is the notation used.

Serials in the above diagram represent the following.

1. This is about when no external padding is used and stride is one(p=0,s=1).
2. This is about when stride is assumed to be 1.
3. This is a general case where nxn input is convolved by kxk kernel with padding p and stride s and the output shape is given.

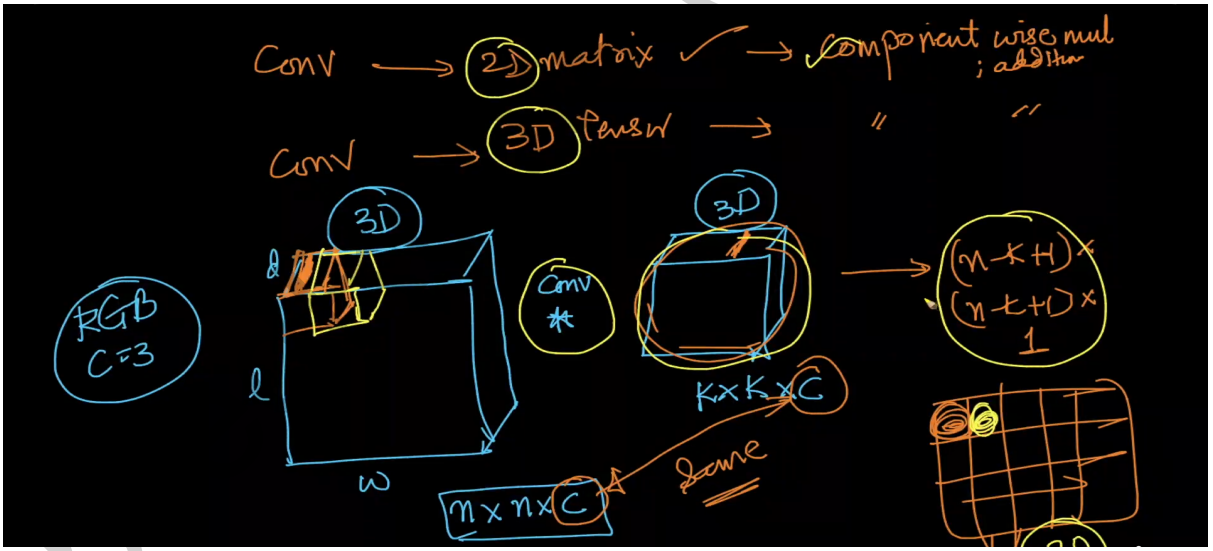
n -> input size nxn
k-> kernel size kxk
s-> stride of (s,s),means that kernel moves by s steps each time it performs a convolution this is by default 1.For example, In python stride 2 on a list L is like writing L[0 : : 2], so we take 2 steps each time.
p-> padding of p rows/columns to the boundary pixels on all the 4 sides of the input. Image size after padding is (n+2p) x (n+2p). Typically 0 is used for padding but many alternatives are also available here.
Floor operator is used here in the formula, y= floor(x) means y is the maximum integer less than or equal to x.

60.4 Convolution over RGB images



Typically a gray scale image has just a single channel but an RGB image has 3 channels. Assume the input be of size nxmxc where c is channels and n,m are length and width of the input. So a color image can be represented as a 3D Tensor.

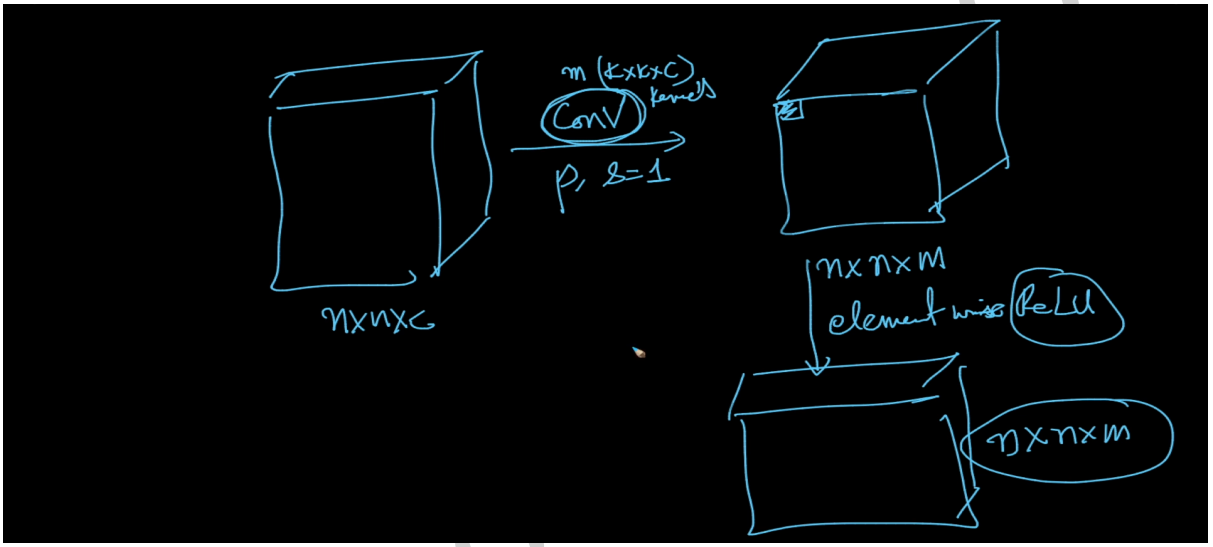
Extending Convolution to RGB Images



Note that the kernel for RGB input will be a 3d Tensor with 3 channels. It helps to imagine the image as a cuboid and kernel also as a smaller cuboid which moves around on the input image convolving its elements.Kernel will have the same number of channels that input has, for the convolution operation to work properly. And the output of kxkxc kernel convolution on nxnxc input image is a 2d array of shape (n-k+1) x (n-k+1).

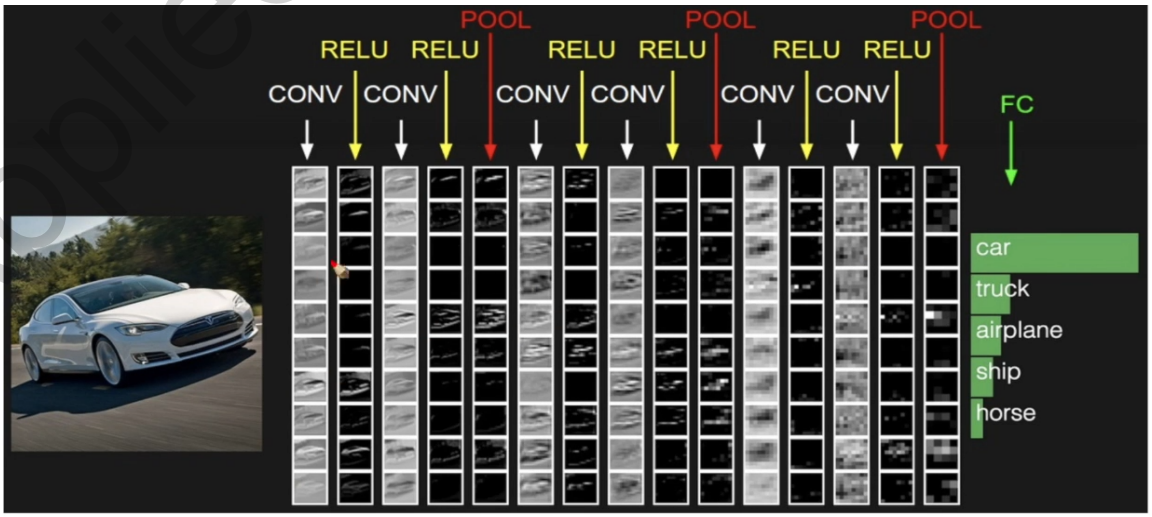
60.5 Convolutional Layer

Kernels are used here as an inspiration from the visual cortex. But kernels are learned here instead of using specific kernels(like sobel) to do the job. Just like we learn weights in an MLP, we learn kernels in a CNN. The output size of convolution depends upon padding,stride,kernel size, but the number of channels in the output depends upon the number of kernels the input is convolved upon. After the convolution operation a non linear activation(like Relu) is introduced just like in MLP.

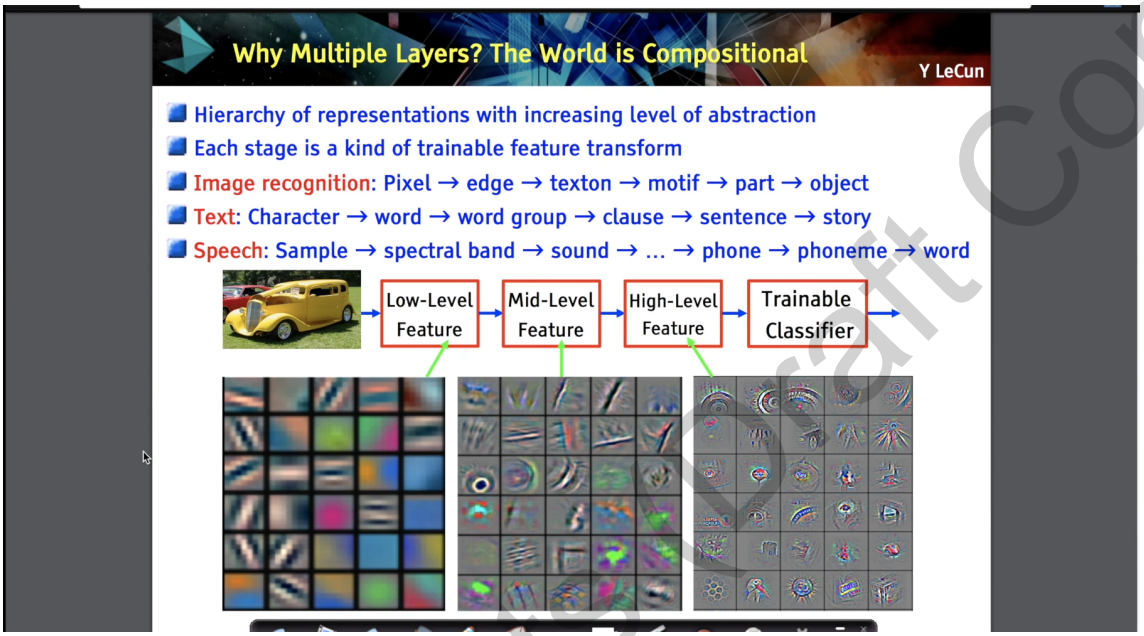


So, m in the output shape nxnxm comes from the number of kernels used to convolve on the input. And padding,stride,number of kernels,kernel size are all hyperparameters.

Convolution Layers stacked on top of each other for image classification.



Just like as found out by hubel and wiesel findings, convolutions also exhibit a kind of hierarchy, this can be clearly seen when we can look at their kernels, They produce low level edge detector type kernels in earlier layers, and the complexity of the shapes they produce, grows gradually till the later layers, with last layers producing very intricate shapes as shown below.

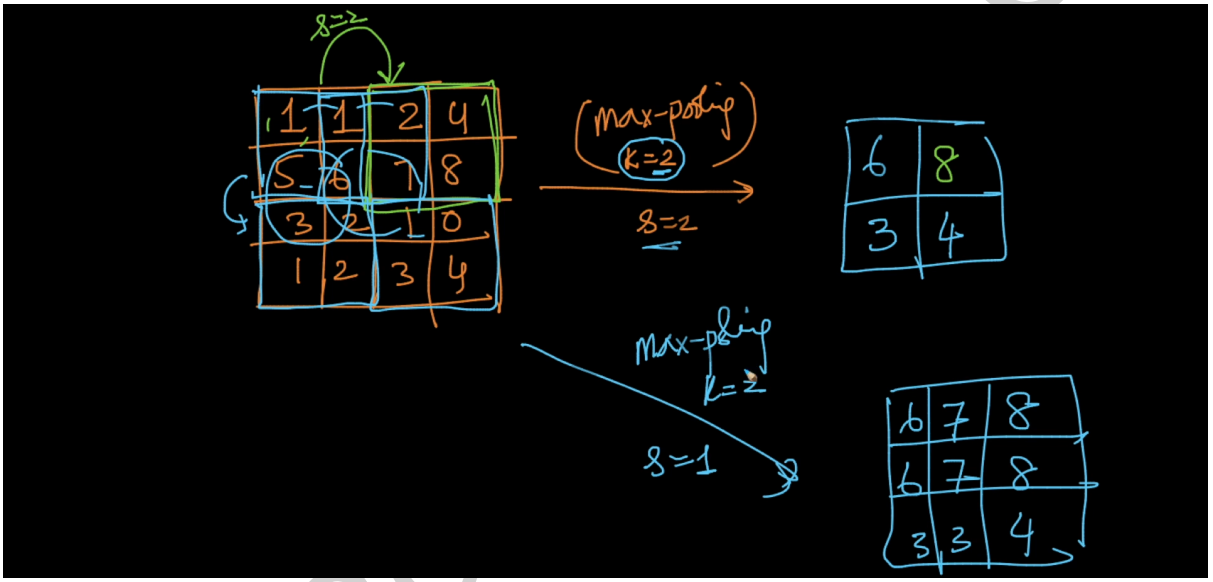


60.6 Max-Pooling

Max-Pooling helps to achieve the following:

- Location Invariance - detect objects irrespective of their location in the input image.
- Scale Invariance - detect even the scaled up or scaled down versions of objects in the image.
- Rotational Invariance - detect objects even with a slight tilt in the image.

Following image shows working of maxpool layer with k= 2 on 4x4 input

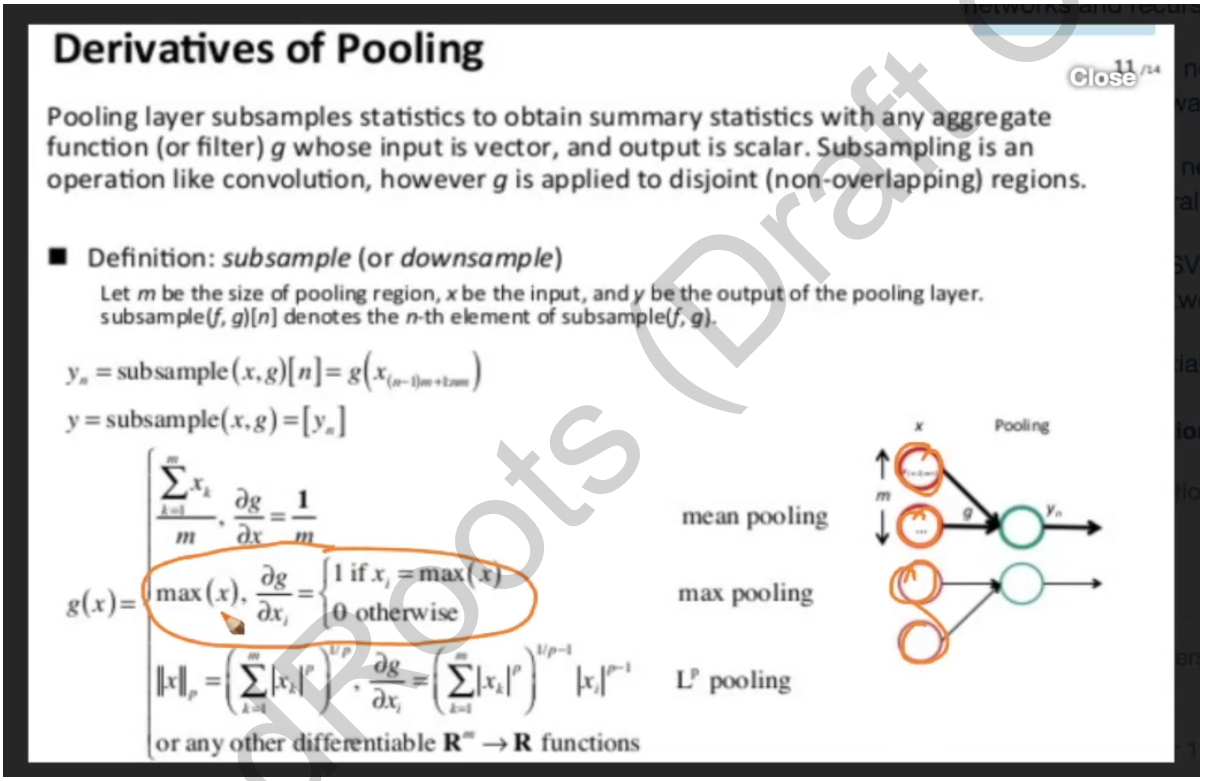


This can be demonstrated by an example of face in any of the regions, assume that in the input above one of the regions has a face so it has a higher number than neighbors, now after max pooling is done, those regions will come out irrespective of where they are, which means that the information about where the face can be used by other layers very well now. This is how Max pooling will play a part in a model architecture. Avg pooling is similar to Maxpool but instead of maximum we take average of the kernel inputs. But Maxpool is much more widely used than Average Pool layer.

60.7 CNN Training: Optimization

Convolutional layer and Pooling layer are only the things that are done differently in CNN Training compared to MLP training. So if these are differentiable operations then we can be sure that Backprop can be done on CNN just like it is done for MLP. As seen in previous lectures, convolution operation is just a generalized dot product. So it is differentiable.

Pooling operations are also differentiable, and they are done as follows.



This means that only the max element in the max kernel will have a derivative 1, rest elements(non-max) will have a derivative 0. This happens because the non-max elements in a maxpool layer will not contribute to the output only max elements do. Similarly in Avg Pooling output is based on all the outputs equally, so all their derivatives will be equal and non-zero(1/m).

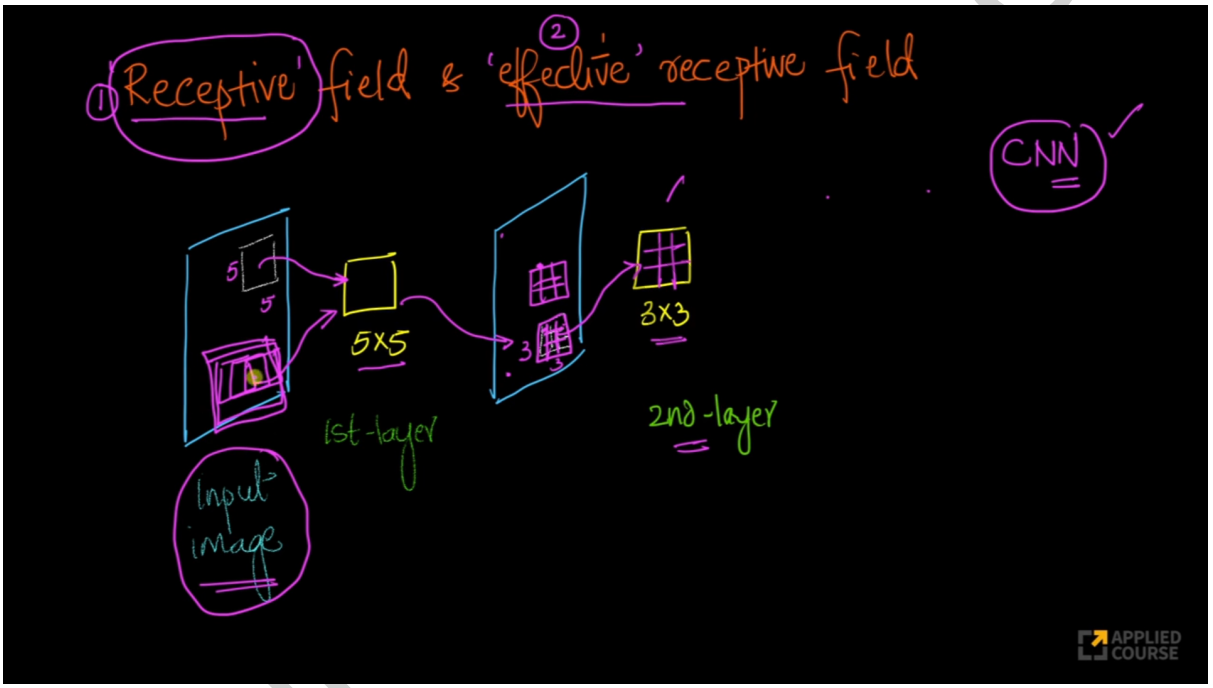
60.8 Receptive Fields and Effective Receptive Fields

Receptive Field:

That part of the image or activation, with which the kernel is being convoluted with.This is the same as kernel size.

Effective Receptive Field:

That part of the **original input**, with which the kernel is being convoluted with indirectly or directly.



Now for the above given image,Receptive field, Effective Receptive field of 5x5 will be 5x5.Receptive field of the 3x3 kernel will be 3x3, and its Effective Receptive field will be 11x11, it means 3x3 kernel indirectly convolutes upon 11x11 size part of the input image. This can be calculated as (3+(5-1)+(5-1)) x (3+(5-1)+(5-1)), here 5 comes from 5x5 kernel and 3 comes from 3x3 kernel.

60.9 Example CNN: LeNet [1998]

LeNet, 1998

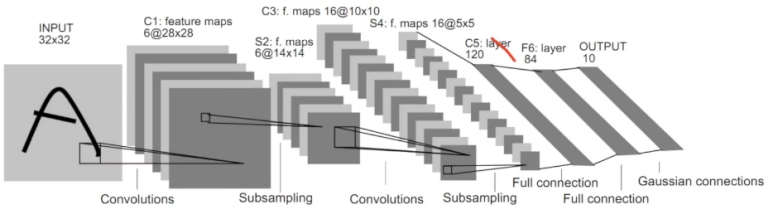


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

This is created by Yann Lecun, one of the founding fathers of Deep Learning.Lenet is a 10 class classifier to classify digits.Subsampling here is same as Mean Pooling with kernel size 2.
Sigmoid Activation has been used here, as relu wasn't invented then.
Input images are grayscale.

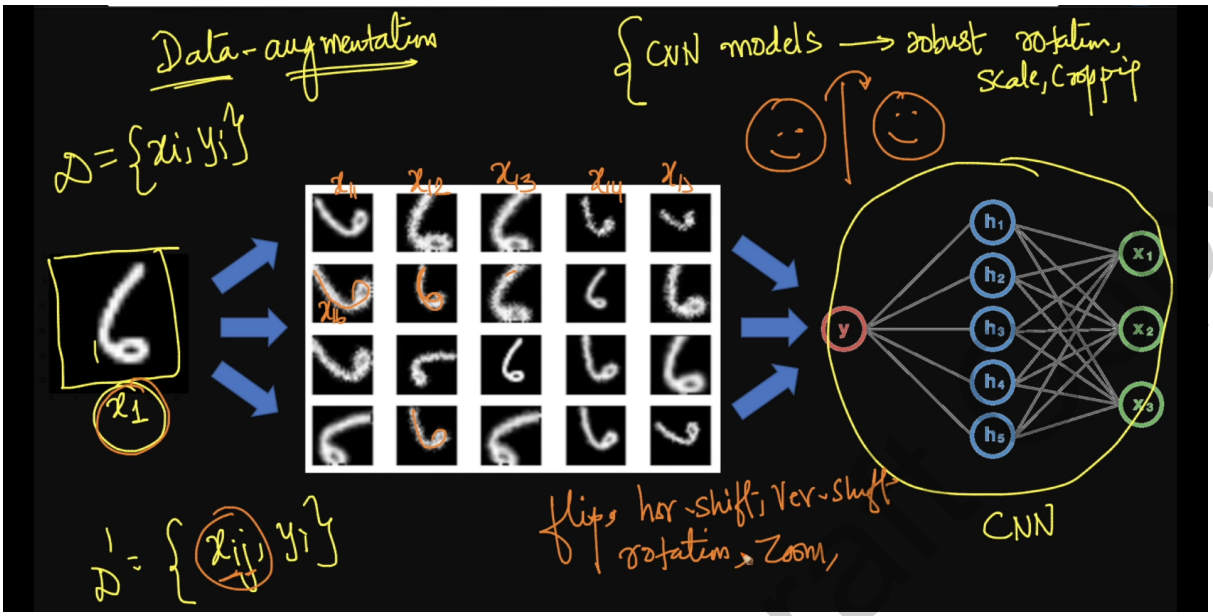
Pls refer to link for more on this, for any help send a mail to mentors.diploma@appliedroots.com

60.10 ImageNet dataset

Very popular dataset with 14 million images with 20,000 ambiguous classes of which, a shortened version with 1000 classes will typically be used.

ILSVRC(ImageNet Large Scale Visual Recognition competition) challenge is based on this dataset. Seminal paper in 2012 Alexnet's results on this dataset, played a big part in making deep learning as popular as it is today. Benchmark dataset to showcase the results.

60.11 Data Augmentation



Data Augmentation on an input image with number 6 in MNIST Dataset.

Data Augmentation helps in increasing the size of the dataset.
Another advantage of this is by training on different variations of the input, the model becomes invariant to these variations, i.e In the above example the model trained on such type of data can classify noisy 6, slightly tilted 6, shifted 6 as 6 much better than a model which was just trained on data without any augmentations.

Other Augmentations that can be done include flipping of the image(left to right or up and down), Zooming in or Zoomed out of the image, Stretching of the image.

60.12 Convolutional Layers in Keras

Conv2D(filters=20,kernel_size=5,strides=1,padding="same")

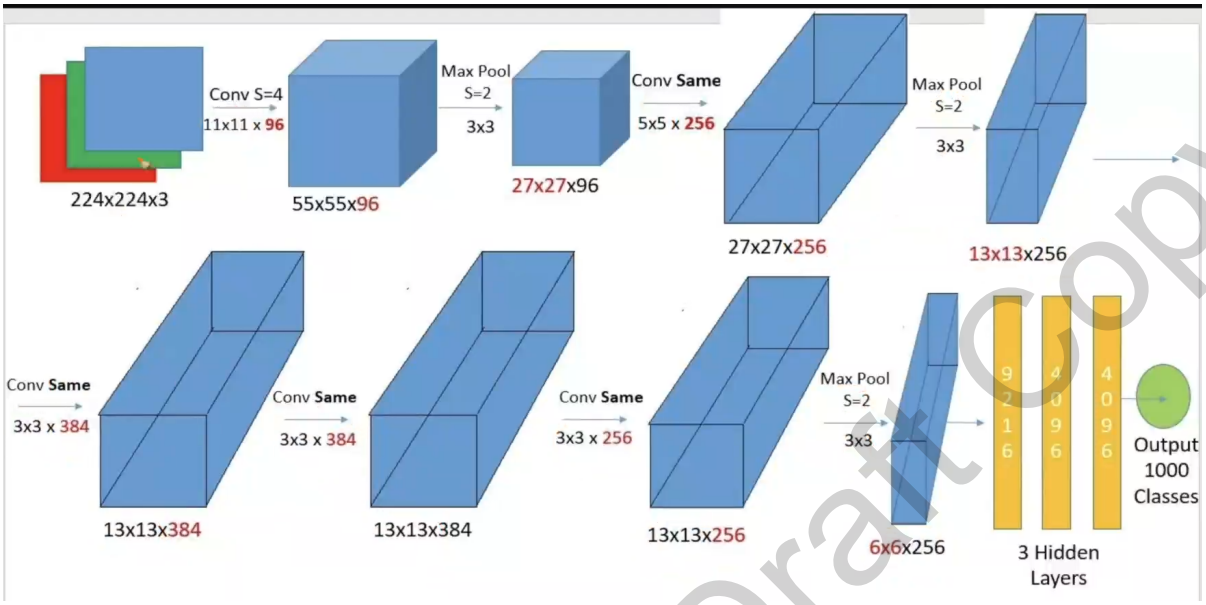
Filters-> the number of output channels(filters)
kernel_size-> Size of the kernel to be used for convolution
strides-> strides
padding-> ‘same’ says that pad such that output shape is same as input shape. ‘valid’ says that no padding needs to be done.
Border_mode is just another name for the padding parameter.

Flatten operation converts a tensor(2d or 3d or nd) to a 1d array. For a 2d array this can be thought of as entering each value from the array, row after row such that the output is of 1D shape. This is used here as the output from the convolution can’t be directly given to the final dense layers so we flatten this output to feed it to dense layers.

These are some commonly used parameters in Conv2D, for more refer here. Since this is code related lecture, for any queries drop a mail to mentors.diploma@appliedroots.com.

60.13 AlexNet

AlexNet is a seminal paper that came out in 2012.



Height x Width x Channels is the representation of shapes shown above.

Conv Same means to add padding such that after convolution operation on the input, the size(length,width) of the output is of the same size as the input. It has used ReLU, dropout, GPU, data augmentation.Local Response Normalization(LRN) is also used here but alternatives for this have been found now for this idea,This is briefly normalization across channels.

```
model = Sequential()

# Layer 1
model.add(Convolution2D(96, 11, 11, input_shape = (1,28,28), border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 2
model.add(Convolution2D(256, 5, 5, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 3
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, border_mode='same'))
model.add(Activation('relu'))

# Layer 4
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(1024, 3, 3, border_mode='same'))
model.add(Activation('relu'))

# Layer 5
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(1024, 3, 3, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 6
model.add(Flatten())
model.add(Dense(3072, init='glorot_normal'))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```

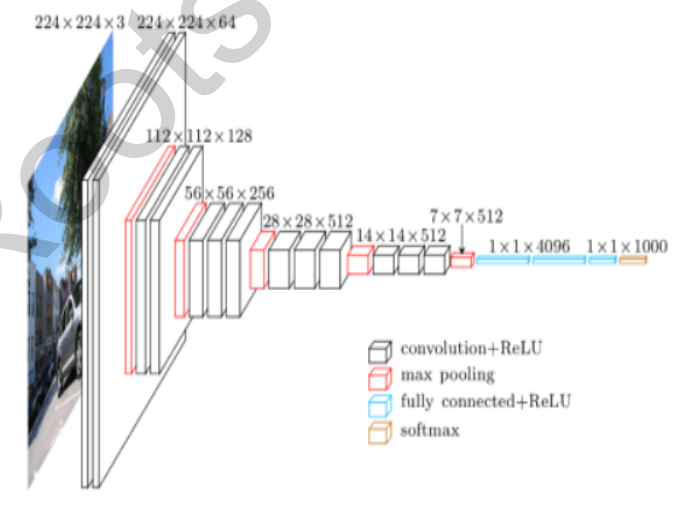
Snippet of code for Alexnet in keras.

60.14 VGGNet

Here only 3x3 kernels are used for all the convolution operation and every max pool is with a kernel size 2x2 and stride 2. This choice greatly simplifies the architecture of the model.

VGG 16 has 16 layers and fewer parameters than VGG 19 which has 19 layers.

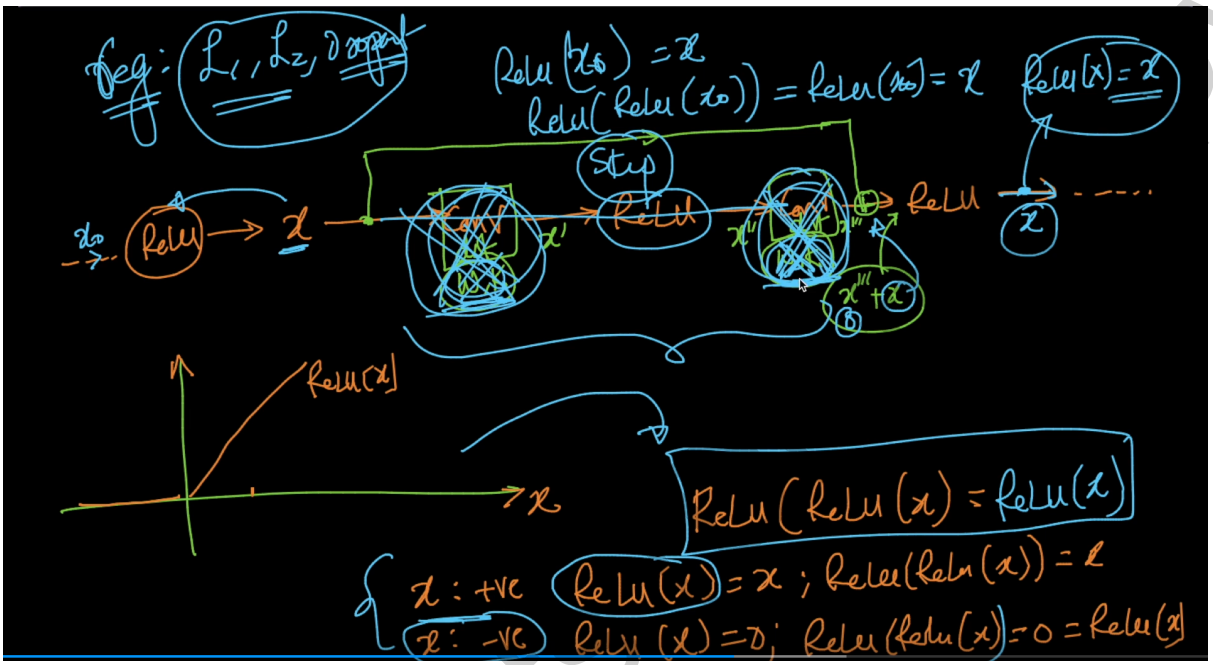
VGG 16 architecture



Keras provides a VGG model built in, the weights of VGG after training on ImageNet dataset are also available, so there's no need to train the model from scratch.

60.15 Residual Network

In models before ResNet, models with a lot of layers used to perform worse than models with lesser layers. This issue forms the basis for Residual Connections.



Residual connection can skip over the useless layers. Think of the crossed out conv layers as useless layers, because skip connection is present and assume regularization is present,then weights of these useless layers will be 0(x”’) but due to skip connection output of will be Relu(x”’+x) = Relu(x). So x has skipped the useless layers which means that additional layers will not hurt performance. If the crossed out layers are useful then x”’ won’t be zero and they will help in improving performance.

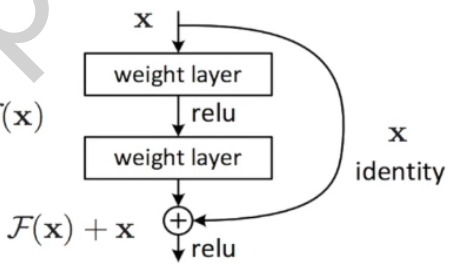
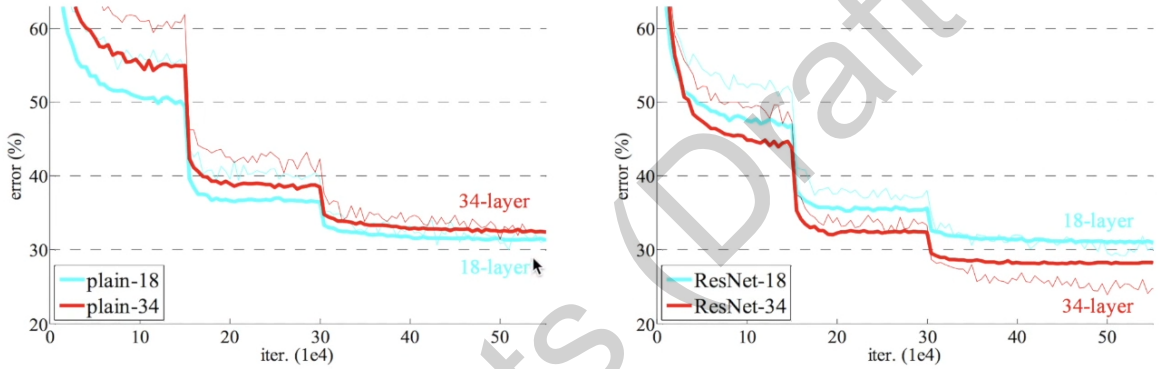


Figure 2. Residual learning: a building block.

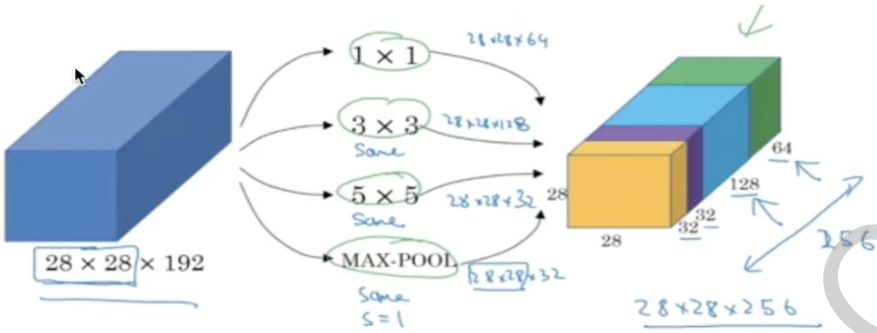
Relu(Relu(x)) = Relu(x) so applying Relu multiple times is the same as applying it once. Skip connections are added just before Relu activation of the conv layers shown above. Here we also should make sure that F(X) is of the same shape as x so that F(x)+x is feasible.

So after this residual connection is implemented in the model, the deeper models are guaranteed to give better performance than comparatively shallow models. This can be demonstrated in the below diagram which compares a model with and without Residual connections.



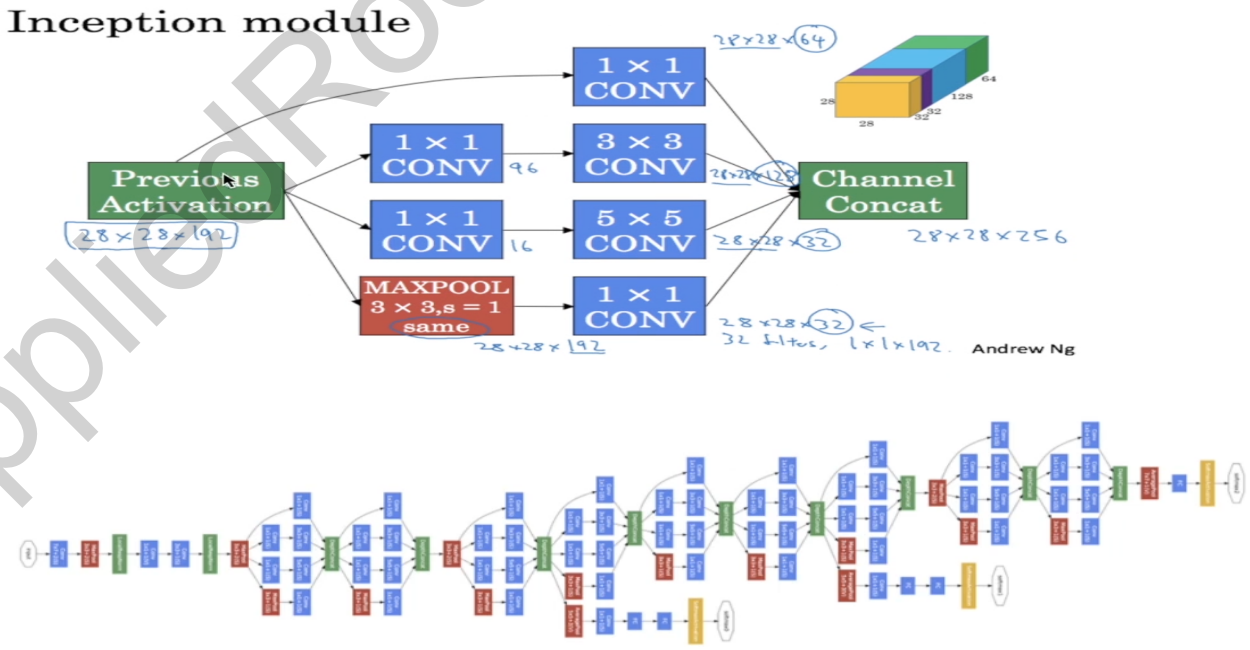
60.16 Inception Network

Instead of choosing whether to use 1x1 or 3x3 or 5x5 kernel sizes or Pooling layers, we use all of them.



Convolution Operation is very expensive, Inception also reduces the computation required, by introducing a bottleneck layer which uses 1x1 convolution to do this.This can be examined by using a 5x5 kernel(32 channels) to directly on 28x28x192, now compare this with using 1x1 kernel(16 channels) first on input with 28x28x192 shape and then on this, use the same 5x5 kernel with 32 channels, Computation is decreased by 10x in the second operation.

Below is the inception module. Notice that 1x1 Conv is used before 3x3 and 5x5.



Notice that the inception module forms the building block to the Inception Network.

60.17 Transfer Learning

Transfer learning means to reuse a pretrained model on the current task instead of building the model completely from scratch.

Suppose for a model pre-trained on ImageNet for Classification, we can remove the few top layers(at the Flatten Layer and mostly dense) and then use the model to predict on the input, then the output we get is known as Bottleneck features.

Freezing the initial layers means no gradient passes through them,so these initial layer's weights won't be updated at all.Fine Tuning means to use lower learning rate to update the weights, this is done so that the weights don't change too drastically.

Advice on Using Transfer Learning

Case 1: Drop the top layers of the model, then use the bottleneck features of the input and then build a model with these bottleneck features.
Case 2: Freeze the initial layer weights and finetune the later layers of the model.
Case 3: Use the pretrained model weights to initialize the model and finetune the complete model.

If Data is **small** in size and **similar** to Imagenet:
Use Case 1

If Data is **medium** in size and **similar** to Imagenet:
Use Case 2

If Data is **large** in size and **similar** to Imagenet:
Use Case 3

If Data is **small** in size and **not similar** to Imagenet:
Use only the earlier layer weights to find the Bottleneck features, then build a model with these features as inputs. This is similar to Case 1

If Data is **large** and **not similar** to Imagenet:
Use Case 3.

Note that we avoid training the model from scratch even when the data is completely different.

Since **60.18,60.19,60.20** are code related lecture, for any queries related to them pls drop a mail to mentors.diploma@appliedroots.com.