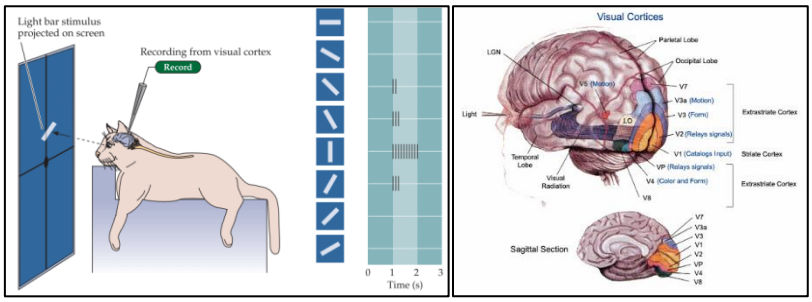
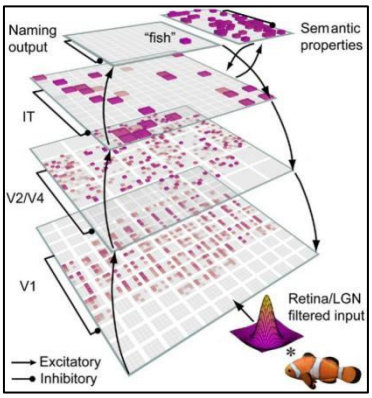
- **Biological inspiration: Visual Cortex**
  - A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and can differentiate one from the other. Used for visual tasks like face recognition, object recognition, etc.
  - In 1959, for an experiment Hubel and Wiesel inserted a micro electrode into the primary visual cortex of an anaesthetised cat. They then projected patterns of light and dark on a screen in front of the cat.
  - They found that some neurons fired rapidly for lines at one angle, while others responded to another angle. Some neurons responded to light patterns and dark patterns differently.



  - The visual cortex is mainly responsible for processing of visual information.
  - It receives neuronal signals from the thalamus and pass these signals through a pipeline that refines and integrates the signals to produce high level concepts.
  - The first stage of this pipeline is called V1 or Visual area1. This region receives the raw neuronal signals from the thalamus. As V1 is early stage in the pipeline, it is sensitive to low abstract features in the image.
  - V1 contains millions of neurons that fire together in distinct patterns to encode the information about the received signals from thalamus of different stimuli which is known in neuroscience as population code.
    - V1 (striate cortex) Separates the information into blobs (colour) and inter blobs (form and movement).
    - V2 Gets input from V1. Consists of stripes: Thick ones (movement), thin ones (colour) and pale zones (form).
    - V3 Gets input from V2. Processes dynamic form.
    - V4 Gets input from V2. Processes colour and form. Its cells are responsive to different perceived colour, with the center being excited by a certain colour, and the surround being inhibited. This plays a role in colour constancy.
    - V5 Gets input from V2. Processes motion.
  - The image processing units are built in a similar way



- **Convolution: Edge Detection on images**
  - For a 6×6-pixel grayscale image. we will have a 6×6×1 tensor & for a colored image 6×6×3.

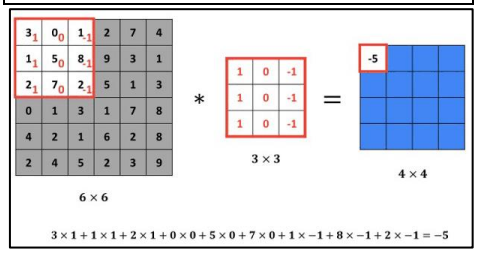
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

  - An edge can be detected by change in colors. To detect this edge, we do convolution operation.
  - The convolution operator uses 3×3 kernel operation which are convolved with the original image to calculate approximations of the derivatives for horizontal/vertical changes.

1	2	1
0	0	0
-1	-2	-1

- Convolution is a dot product between 3x3 kernel matrix and every possible 3x3 matrix from original 6x6 matrix.

$input$ $n\ X\ n$	$(k\ X\ k)$ $\rightarrow$	$output$ $(n-k+1)\ X\ (n-k+1)$
$6\ X\ 6$	$(3\ X\ 3)$ $\rightarrow$	$(6-3+1)\ X\ (6-3+1)$ $4\ X\ 4$



- After doing this convolution operation, we get a 4x4 matrix as follows

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

- To represent the above matrix in form of image all values should be in a range of 0 to 255. We need to re normalize above vector
  - $S_{min} = 0;\ S_{max} = 255$ range we want to bring our values to.
  - $X_{min} = -1020;\ X_{max} = 0$ current pixel maximum and minimum range
  - $x'(converted\ value) = \frac{(x - Xmin)*(Smax - Smin)}{(Xmax - Xmin)} = \frac{(x + 1020)*(255)}{(1020)}$
  - when x = 0,
  - $x' = \frac{(0 + 1020)*255}{1020} = 255$
  - when x = -1020,
  - $x' = \frac{(-1020 + 1020)*255}{1020} = 0$

255	255	255	255
0	0	0	0
0	0	0	0
255	255	255	255

- Basically, it represents original vector into a new vector with smaller dimensions such that the edges in the original image will be represented as two rows
- The above edge detector was horizontal edge detector similarly there is a vertical one which also has same procedure & sobel vertical as follows

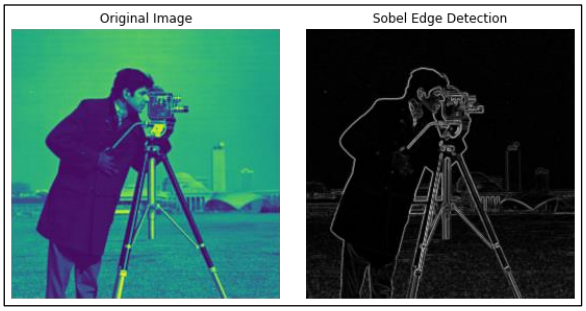
1	0	-1
2	0	-2
1	0	-1

- If we apply same convolution to detect vertical edges such that 6x6 matrix is as follows

0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255

- The results will be

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255



- **Convolution: Padding**
  - As seen earlier after convolution the 6 x 6 matrix was converted into 4 x 4 matrix. If I wish to have 6 x 6 matrix even after convolution, we need to perform padding.

$$\underset{n \times n}{input} \quad \underset{\rightarrow}{(k \times k)} \quad \underset{(n-k+1) \times (n-k+1)}{output}$$

  - In the above formula we want $n - k + 1 = 6$. we can't change k hence make $n = 8$. The original matrix will be updated with an additional padding on outside to reform it to 8x8

	255	255	255	255	255	255	
	255	255	255	255	255	255	
	255	255	255	255	255	255	
	0	0	0	0	0	0	
	0	0	0	0	0	0	
	0	0	0	0	0	0	

  - Zero padding: all padding values will be filled with 0. This will give some additional edges to the image. This is used extensively in CNN.

0	0	0	0	0	0	0	0
0	255	255	255	255	255	255	0
0	255	255	255	255	255	255	0
0	255	255	255	255	255	255	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

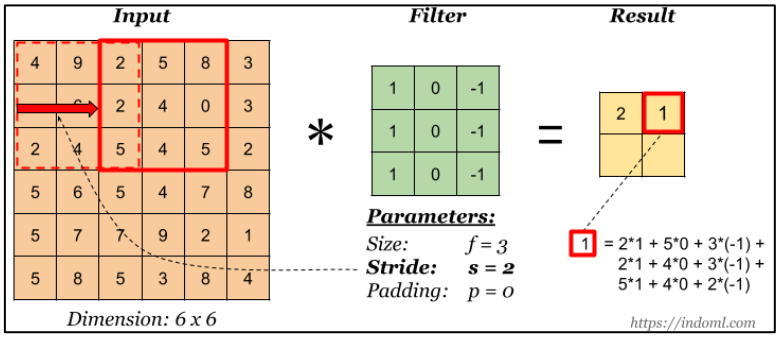
  - Same value padding: all padding values will be filled with value from nearby cell.

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

  - Here the original formula will be changed to

$$\underset{n \times n}{input} \quad \underset{\rightarrow}{padding} \quad \underset{(n+2p) \times (n+2p)}{padded\ input} \quad \underset{\rightarrow}{(k \times k)} \quad \underset{(n+2p-k+1) \times (n+2p-k+1)}{output}$$

- **Convolution: Strides**

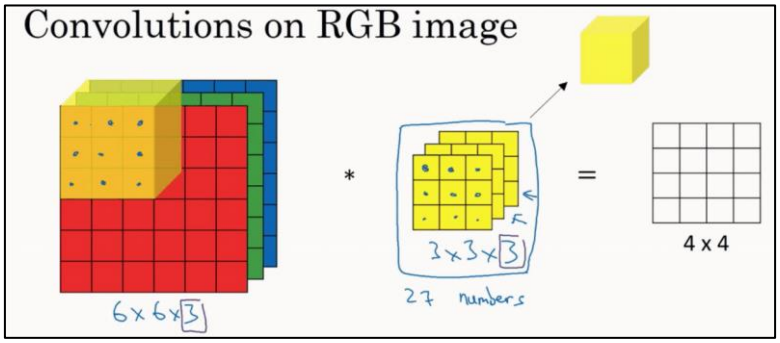


  - Stride is a parameter that modifies the amount of movement over the image. E.g., if a neural network's Stride is set to 1, the filter will move one pixel
  - The size of the filter affects the encoded output volume, so Stride is often set to a whole integer, rather than a fraction or decimal.
  - Stride is useful for reducing size of input dramatically

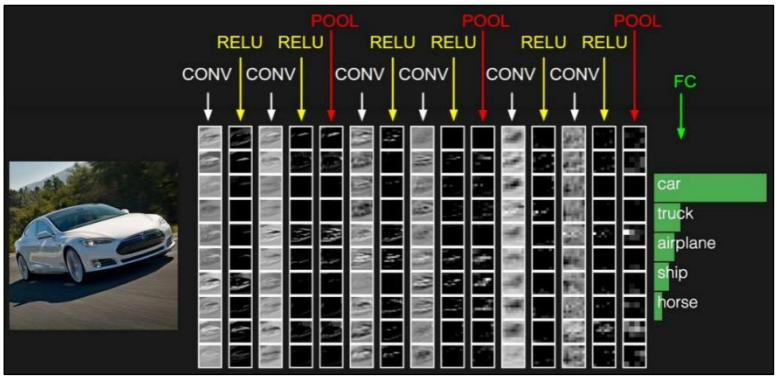
$$\underset{n\ X\ n}{} \quad \underset{k \times k}{\overset{s=j}{\rightarrow}} \quad \left(\left\lfloor \frac{n-k}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{n-k}{s} \right\rfloor + 1\right)$$

- **Convolution over RGB images**
  - In a grayscale image each cell is a 1D matrix representing values from 0 to 255. In case of color images each cell represents a 3 features R, G, B.

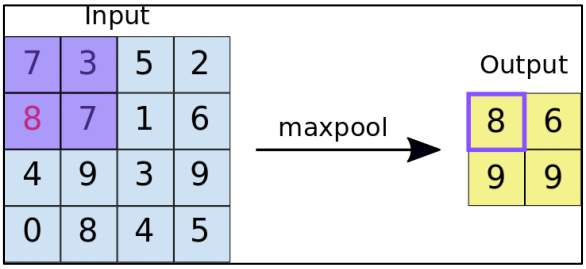
- An alternate way to interpret is colour image is a combination of 3 images. A colour image can be represented as 3D tensor of size $n \times m \times c$ $(L \times B \times D)$
- The convolution procedure will be like that of 2D. but c (depth) should be same in case of both tensors



- **How CNN learn?**
  - Convolutional layers are inspired by visual cortex which has multiple visual areas v1, v2, v3, etc. In v1 itself there are multiple edge detectors which are represented as multiple kernels in CNN.
  - MLP learns weights by using backpropagation. In CNN these kernels will be learnt by backpropagation
  - A CNN layer will take input image ($n \times n \times c$) then pass it through multiple m kernels ($k \times k$) with enough padding & Stride and gives an output m $n \times n$ matrix.
  - For e.g., an $16 \times 16 \times 3$ image passed through 10 kernels of size $3 \times 3 \times 3$ will give us 10 images of size 16 x 16 with each image will have some different feature detection such as horizontal edges, vertical edges, etc.
  - Hyper parameters to tune are: no of kernels, padding, Stride
  - The typical convolution procedure is we have an image of size $n \times n \times c$ which gives an output of $n \times n \times m$ after convolution. On top of this output element wise ReLU function is performed

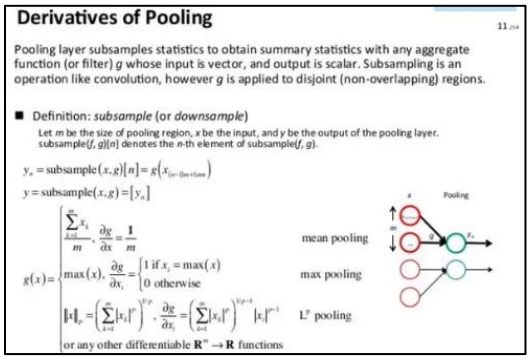


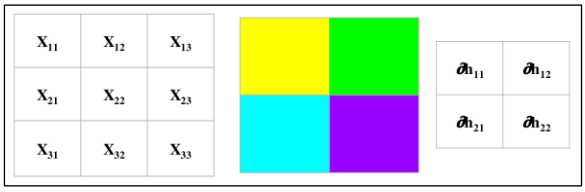
- **Max-pooling**
  - A small set of neurons fire when we see invariance such as location invariance, scale invariance, rotational invariance in image. This is the inspiration for Max pooling in CNN.
  - Max pooling is like kernelization such that the original matrix is reduced to different dimension as follows. Following Max pooling has size of $k = 2$ & $S = 2$. $k = 2$ means the window size is of 2 x 2 & S =2 means we are jumping with this window with a gap of 2 columns.
  - The intuition is if we see an image it may have a face at 2 distant locations, Max pooling will give output vector which retains the information from original huge matrix but now the matrix will be smaller.
  - Subsequently another layer of Max pooling can be performed on this first one.



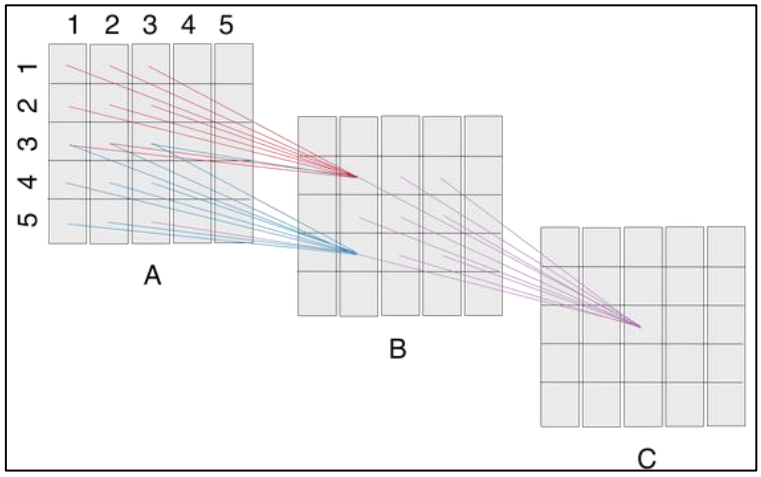
- **CNN Training: Optimization**
  - In MLP we had backpropagation followed by optimizers like SGD, Adam, etc. backpropagation was basically finding weights to compute $\hat{y}$ such that loss is minimized
  - For optimization the major requirement was to have a propagation & activation function to be differentiable.
  - In case of CNN, we have convolution function followed by ReLU.

- ReLU is differentiable & also the convolution function is dot product. In case of MLP we were doing this over vectors & in CNN we are doing it over Matrices.
- The derivative of Max pooling layer will be calculated such that

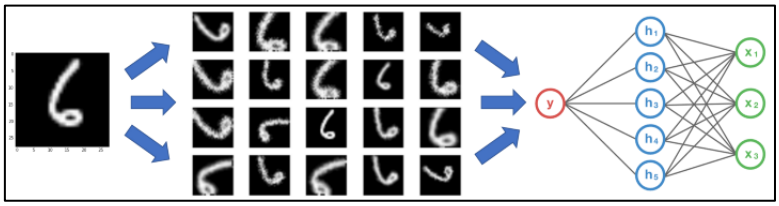




- **Receptive Fields and Effective Receptive Fields**
  - Receptive field is that part of image on which the convolution kernel operates on at a given point of time.
  - Effective receptive field is the area of the original image that is being indirectly processed by later convolutions after second layer.

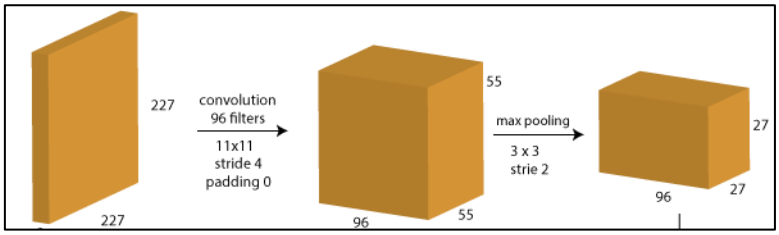


  - In above example, The A, B and C are convolutional layers of a CNN(with padding & S=0,K=3x3)
  - Receptive field of C(3,3) is B(2:4, 2:4) and effective receptive field is A(1:5,1:5)
- **Data Augmentation**

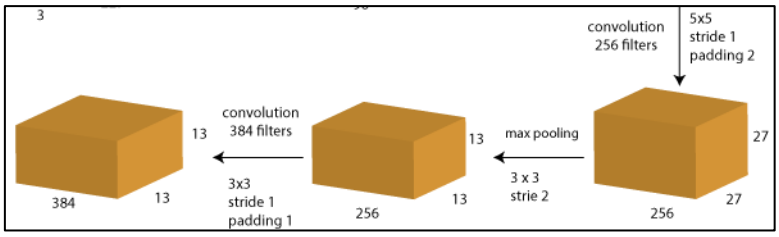


  - A CNN with invariance property can robustly classify objects even if it's placed in different orientations such as invariant to translation, viewpoint, size, or illumination (Or a combination of the above).
  - This is the idea of data augmentation. We may have a dataset of images with limited set of conditions. But query images may exist in a variety of conditions e.g., with different orientation, location, scale, brightness etc.
  - The original images are modified by flipping, horizontal shift, vertical shift, sheer, rotation, zoom etc. The target variable for each image will remain same

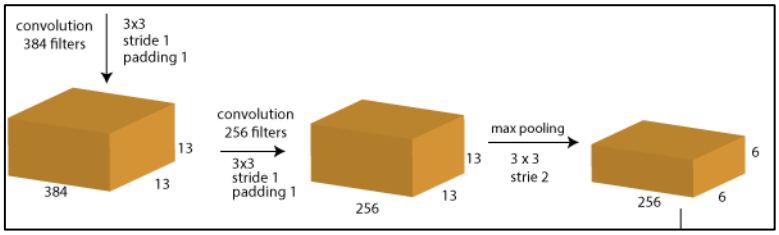
- **Alex net**



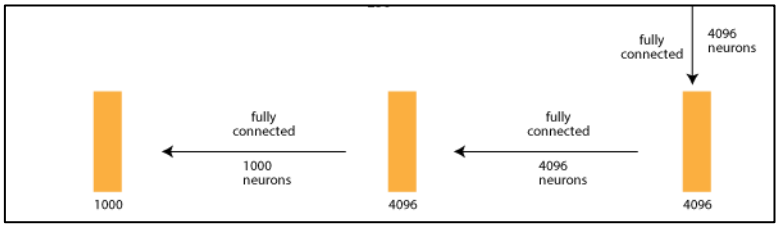
  - Input Image: $n = 227 \times 227 \times 3$
  - **Conv1**: $k = 11 \times 11, m = 96, S = 4.$
    Output = $\left\lfloor \frac{n-k}{s} \right\rfloor + 1 = \left\lfloor \frac{227-11}{4} \right\rfloor + 1 = 55 \times 55 \times 96$
  - **Maxpool1**: $k = 3 \times 3, S = 2$
    Output = $\left\lfloor \frac{55-3}{2} \right\rfloor + 1 = 27 \times 27 \times 96$



  - **Conv2:** $k = 5 \times 5, m = 256, S = 1, Padding = 2$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{27+2*2-5}{1} \right\rfloor + 1 = 27 \times 27 \times 256$
  - **Maxpool2**: $k = 3 \times 3, S = 2$
    Output = $\left\lfloor \frac{27-3}{2} \right\rfloor + 1 = 13 \times 13 \times 96$
  - **Conv3**: $k = 3 \times 3, m = 384, S = 1, P = 2.$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{13+2*1-3}{1} \right\rfloor + 1 = 13 \times 13 \times 384$

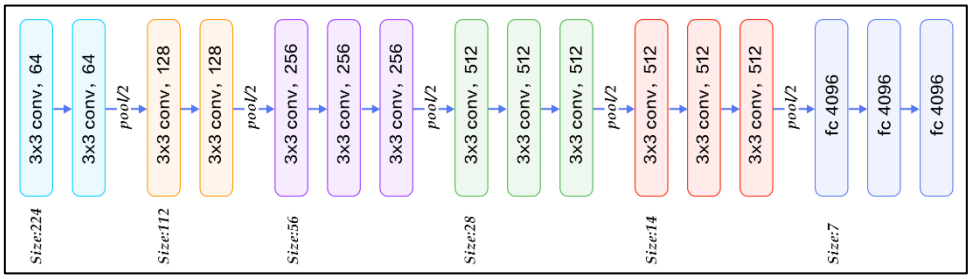


  - **Conv4**: $k = 3 \times 3, m = 384, S = 1, P = 2.$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{13+2*1-3}{1} \right\rfloor + 1 = 13 \times 13 \times 384$
  - **Conv5**: $k = 3 \times 3, m = 256, S = 1, P = 2.$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{13+2*1-3}{1} \right\rfloor + 1 = 13 \times 13 \times 256$
  - **Maxpool3**: $k = 3 \times 3, S = 2$
    Output = $\left\lfloor \frac{13-3}{2} \right\rfloor + 1 = 6 \times 6 \times 256 = 9216$
  - We're going to unroll this into 9216 neurons to final fully connected layers.

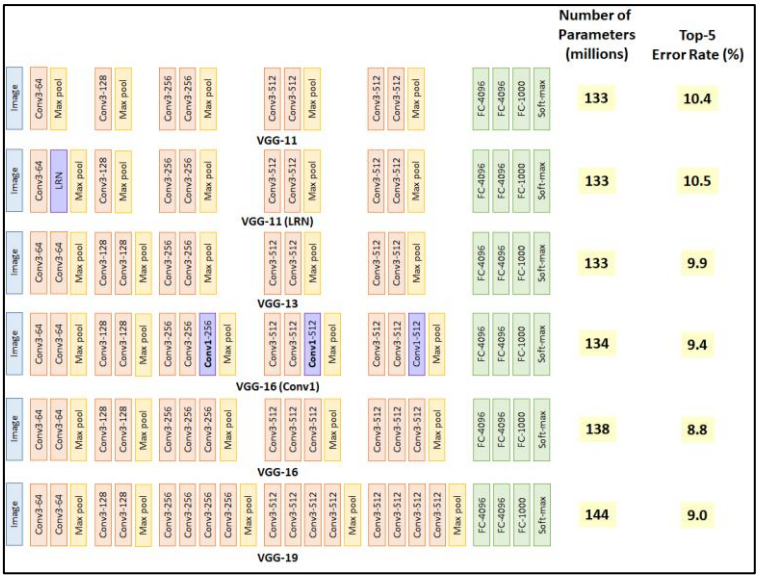


  - **FC6, FC7, Softmax**
  - The first two FC layers have 4096 neurons
  - The final FC layer is SoftMax with 1000 neurons to make classification

- **CNN VGG 16 and VGG 19**
  - In this network smaller filters are used, but the network was built to be deeper than Alex net, LeNet. In VGG−16, instead of having so many hyper parameters we use a much simpler network.



  - Input Image: $n = 224 \times 224 \times 3$
  - **Conv1-1, Conv1-2**: $k = 3 \times 3, m = 64, S = 1, Padding = Same$
    If $Padding = same\ p = \frac{k-1}{2} = \frac{3-1}{2} = 1$ Reference
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{224+2*1-3}{1} \right\rfloor + 1 = 224 \times 224 \times 64$
  - **Maxpool1**: $k = 2 \times 2, S = 2$
    Output = $\left\lfloor \frac{224-2}{2} \right\rfloor + 1 = 112 \times 112 \times 64$
  - **Conv2-1, Conv2-2**: $k = 3 \times 3, m = 128, S = 1, Padding = Same$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{112+2*1-3}{1} \right\rfloor + 1 = 112 \times 112 \times 128$
  - **Maxpool2**: $k = 2 \times 2, S = 2$
    Output = $\left\lfloor \frac{112-2}{2} \right\rfloor + 1 = 56 \times 56 \times 128$
  - **Conv3-1, Conv3-2**: $k = 3 \times 3, m = 256, S = 1, Padding = Same$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{56+2*1-3}{1} \right\rfloor + 1 = 56 \times 56 \times 256$
  - **Maxpool3**: $k = 2 \times 2, S = 2$
    Output = $\left\lfloor \frac{56-2}{2} \right\rfloor + 1 = 28 \times 28 \times 256$
  - **Conv4-1, Conv4-2, Conv4-3**: $k = 3 \times 3, m = 512, S = 1, Padding = Same$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{28+2*1-3}{1} \right\rfloor + 1 = 28 \times 28 \times 512$
  - **Maxpool4**: $k = 2 \times 2, S = 2$
    Output = $\left\lfloor \frac{28-2}{2} \right\rfloor + 1 = 14 \times 14 \times 512$
  - **Conv5-1, Conv5-2, Conv5-3**: $k = 3 \times 3, m = 512, S = 1, Padding = Same$
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{14+2*1-3}{1} \right\rfloor + 1 = 14 \times 14 \times 512$
  - **Maxpool5**: $k = 2 \times 2, S = 2$
    Output = $\left\lfloor \frac{14-2}{2} \right\rfloor + 1 = 7 \times 7 \times 512$
  - **FC6, FC7, FC8, SoftMax**
  - The first three FC layers have 4096 neurons
  - The final FC layer is SoftMax with 1000 neurons to make classification
  - VGG−16 refers that it has 16 layers that have weights. This is a large network where total parameters are 138 million. However, the simplicity of the VGG−16 architecture made it quite appealing. The number of filters is roughly doubling through every stack of conv layer. There are other variants like VGG19,VGG11



- **Residual Network**

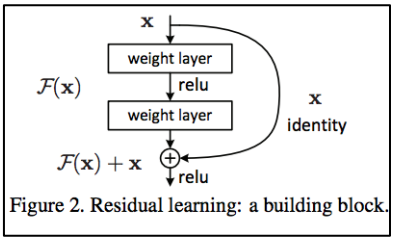
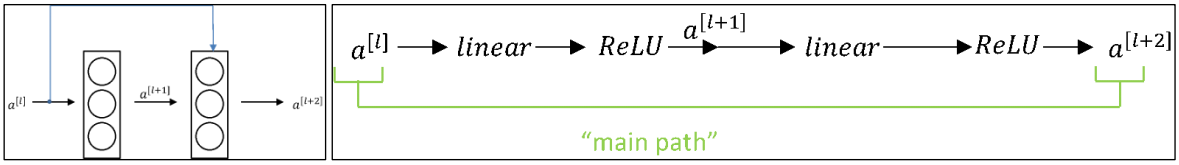


Figure 2. Residual learning: a building block.

  - This identity mapping or skip connection in ResNets adds the output from the previous layer x to the layer ahead $f(x)$. It does not have any parameters.
  - Sometimes x and $f(x)$ will not have the same dimension. The identity mapping is multiplied by a linear projection W to expand the channels of short cut to match the residual.
  - If $f(x)$ becomes zero due to regularization, drop-outs, etc., further activation function will also be zero.
  - If we add x to $f(x)$ skipping the in-between steps, further layers won't decrease the performance.
  - ResNets are built out of a residual block which consists of two layers of a neural network where we start with activation $a[l]$, then we pass it through a residual block and we will get $a[l+2]$



  - $a[l]$ is input and then first linear operator is applied to it

$$z[l+1] = W[l+1] * a[l] + b[l+1]$$

  - Then ReLU non-linearity is applied to get $a[l+1]$

$$a[l+1] = ReLU(z[l+1])$$

  - Then, linear step is applied again

$$z[l+2] = W[l+2] * a[l+1] + b[l+2]$$

  - After applying ReLU function we will get $a[l+2]$

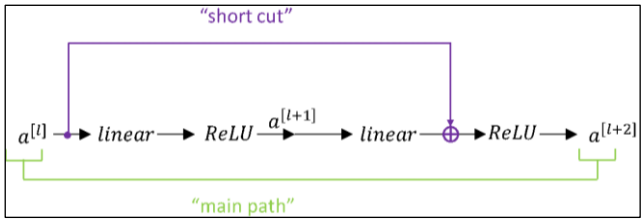
$$a[l+2] = ReLU(z[l+2])$$

  - So, information from $a[l]$ to flow to $a[l+2]$ going through main path of this set of layers

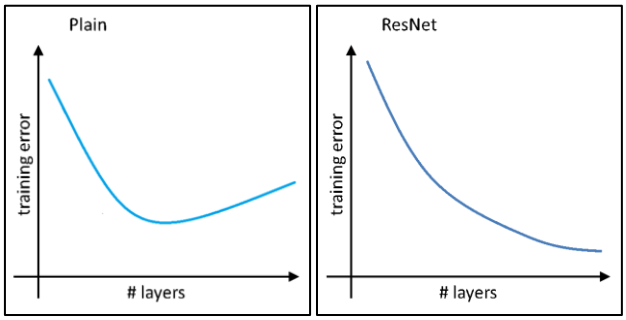
$$a[l] \rightarrow linear \rightarrow ReLU \rightarrow a[l+1] \rightarrow linear \rightarrow ReLU \rightarrow a[l+2]$$

  - To make it a residual block $a[l]$ will be added to $z[l+2]$ and passed through ReLU. This is skip connection or short cut which is used to go much deeper into the neural network.

$$a[l+2] = g(z[l+2] + a[l])$$

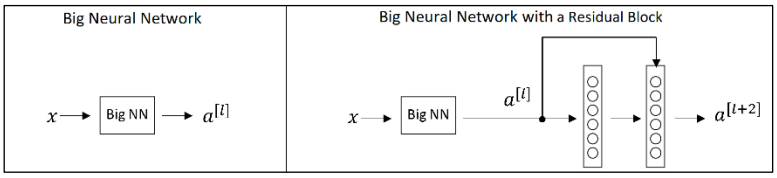


  - Residual blocks allow training of much deeper neural networks. ResNets are built by stacking a lot of residual blocks together.
  - In theory, as we make a neural network deeper, it should only do better and better on the training set. however, in practice a very deep plain network would require much harder time in training. Training error gets worse if we pick a network that's too deep. The training error will tend to decrease after a while but then it'll tend to increase with any standard optimization algorithm

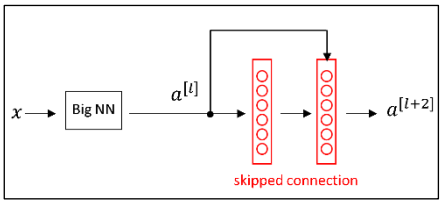


  - ResNets allow us to train much deeper neural networks without a loss in performance. Maybe at some point this graph bellow will plateau and it doesn't help to increase a number of layers.

- **Why ResNets work?**



  - Here we are using the ReLU activation function, so all the activations will be ≥ 0 with exception of the input x., as $X = a[0]$ may not be greater than or equal to 0.
  - Equations for the neural network with a residual block are:
    - ⇨ $a[l+2] = g(z[l+2] + a[l])$
    - ⇨ $a[l+2] = g(W[l+\mathbf{2}]a[l+\mathbf{1}] + b[l+\mathbf{2}] + a[l])$
    - ⇨ If $W[l+\mathbf{2}] = \mathbf{0}$ and $b = \mathbf{0}$ then $a[l+2] = g(a[l]) = a[l]$
  - The identity function is easy for residual block to learn and it's easy to get $a[l+2] = a[l]$ because of a skipped connection. Thus, it's not going to affect model's performance.



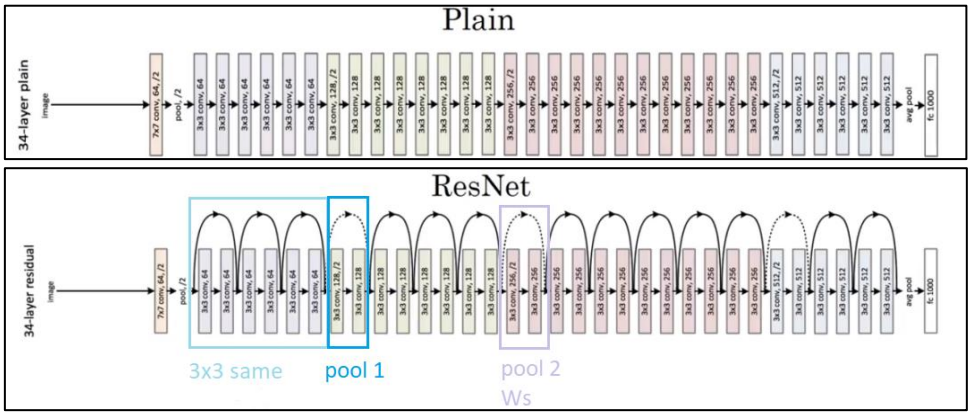
  - It is so easy for those extra layers to learn the residual if it doesn't learn anything in between.
  - The residual networks with the residual blocks:
    - Can learn identity function
    - Do not hurt performance
    - And after a residual block, the gradient descent is capable to improve further on.
  - **What if the input and the output of a residual block have a different dimension?**
  - For a short connection, we are assuming that $z[l+\mathbf{2}]$ and $a[l]$ have the same dimension. In ResNets there are a lot of convolutions with same padding. Therefore,

$$Dim_{res\ block\ input} = Dim_{res\ block\ output}$$

    We are considering the following equation:

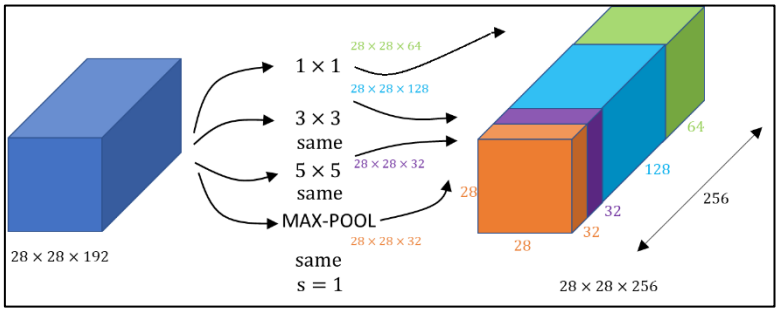
$$a[l+2] = g(W[l+\mathbf{2}]a[l+\mathbf{1}] + b[l+\mathbf{2}] + a[l])$$

    - But if $a[l] \in \mathbb{R}^{128}$ and $a[l+2] \in \mathbb{R}^{256}$, We will add $W_s \in \mathbb{R}^{\mathbf{256\times128}}$
    - Therefore, $(W_s \times a[l]) \in \mathbb{R}^{256}$
    - W_s could be a matrix of parameters to be learned or a matrix that implements zero padding to $a[l]$ and makes it $\mathbb{R}^{256}$
  - ResNets on images
  - To turn plain network into a ResNet we add skipped connections. There are a lot of 3×3 convolutions in ResNet and most of these are convolutions with same padding. Thus, vector addition $z[l+2] + a[l]$ is possible.



  - Similar to other networks, we have a bunch of convolutional layers and then pooling layers and then at the end we have a Fully connected layer that makes a prediction using a SoftMax.

- **Inception Network**
  - In the Inception network, instead of choosing the desired kernel size in a conv layer or choosing between convolutional layer or a pooling layer, we apply all of them.



  - Suppose $n = 28 \times 28 \times 192$.
  - Commonly, we have to choose between conv layer or a pooling layer. Inception network solves this as follows.

    Conv1a: $k = 1 \times 1, m = 64, S = 1$

    Output = $\left\lfloor \frac{n-k}{s} \right\rfloor + 1 = \left\lfloor \frac{28-1}{1} \right\rfloor + 1 = 28 \times 28 \times 64$

    Conv1b: $k = 3 \times 3, m = 128, S = 1, Padding = Same$

    If $Padding = same, p = \frac{k-1}{2} = \frac{3-1}{2} = 1$

    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{28+2*1-3}{1} \right\rfloor + 1 = 28 \times 28 \times 128$

    Conv1b: $k = 5 \times 5, m = 32, S = 1, Padding = Same$

    If $Padding = same, p = \frac{k-1}{2} = \frac{5-1}{2} = 2$

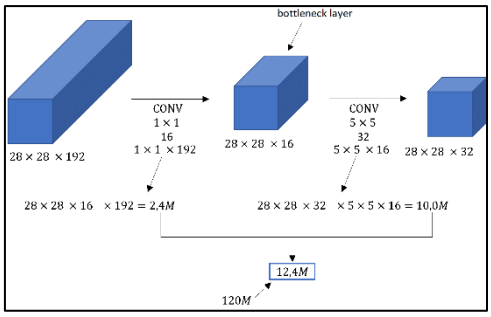
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{28+2*2-4}{1} \right\rfloor + 1 = 28 \times 28 \times 32$

    Maxpool: $k = 3 \times 3, S = 1, Padding = Same$

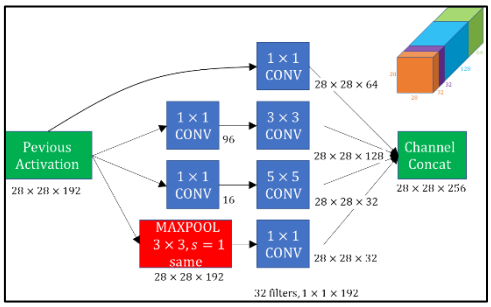
    Output = $\left\lfloor \frac{n+2p-k}{s} \right\rfloor + 1 = \left\lfloor \frac{28+2*1-3}{1} \right\rfloor + 1 = 28 \times 28 \times 192$

    This output is passed through $k = 1 \times 1, m = 32$

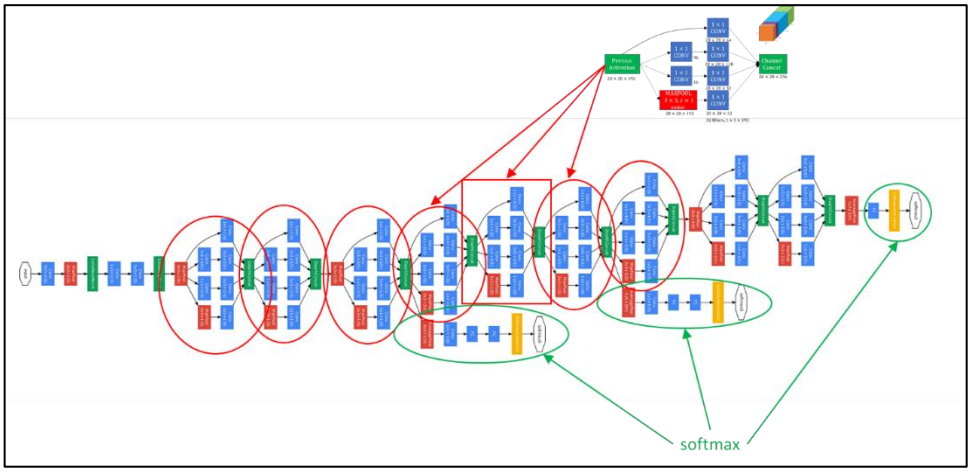
    Output = $\left\lfloor \frac{n-k}{s} \right\rfloor + 1 = \left\lfloor \frac{28-1}{1} \right\rfloor + 1 = 28 \times 28 \times 32$
  - With an inception module like this we can input $\in \mathbb{R}^{192}$ and output will be $\mathbb{R}^{64} + \mathbb{R}^{128} + \mathbb{R}^{32} + \mathbb{R}^{32} = \mathbb{R}^{256}$
  - The problem with inception layer is computational cost.
    - For $k = 5 \times 5$ $m = 32$ where **output** is $28 \times 28 \times 32$ with input $n = 28 \times 28 \times 192$
  - The output size is $28 \times 28 \times 32$ for each $28 \times 28 \times 32$ numbers we need to do $5 \times 5 \times 192$ multiplications.
    - Total number of calculations = $(28 \times 28 \times 32) \times (5 \times 5 \times 192) = 120{,}422{,}400$
  - 120 million multiplies is expensive operation even on modern computer. Using the idea of 1×1 convolution, we'd be able to reduce the computational cost by about a factor of 10.
  - With 1×1 convolution the input of big volume is reduced to $n = 28 \times 28 \times 16$ from $n = 28 \times 28 \times 192$.
  - On this smaller volume 5×5 convolution is applied to get final output of size $28 \times 28 \times 32$ which is same as above. 1×1 convolution is also called a bottleneck layer



  - Total number of calculations = $[(28 \times 28 \times 16) \times (1 \times 1 \times 192)] + [(28 \times 28 \times 32) \times (5 \times 5 \times 16)] = 12{,}443{,}648$
  - This can be summarized as

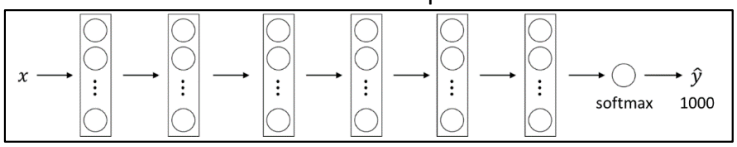


- The inception network consists of a more number of such modules stacked together.

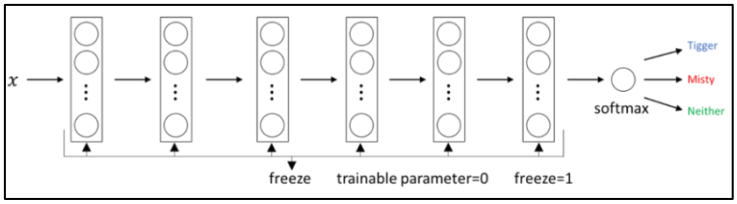


- There are additional side branches which take hidden layer passes it through a few fully connected layers, and then the SoftMax tries to predict what's the output label.
- This is another detail of the inception network which helps to ensure that the features created even at the intermediate layers can predict the output of an image. This have a regularizing effect on the inception network and prevents it from overfitting.

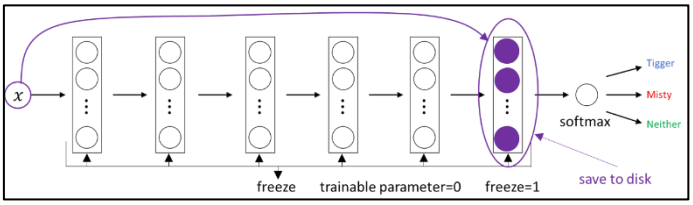
- **Transfer Learning**
  - Many researchers have trained their algorithms on popular datasets. Sometimes this training takes weeks and might take many GPUs.Rather than training a neural network from scratch, we make much faster progress if we download pre trained network's weights and we can use that to a new task that we are solving.
  - We can use them as a parameter initialization for own neural network. i.e., **we can use transfer learning to transfer knowledge from these very large public datasets to our own problem**.
  - For a classification problem with three classes. Algorithm needs to decide if the picture is *Tigger cat*, *Misty cat* or *Neither*. We don't have a lot of pictures of *Tigger* or *Misty* so **our training set is small**. A solution is to download some open-source implementation of a neural network, and download both the code and the weights.
  - Imagenet dataset has 100 classes and one of the class is cat. This data will be helpful to us. The pre trained network will have its own SoftMax unit that outputs one of the 1000 classes.



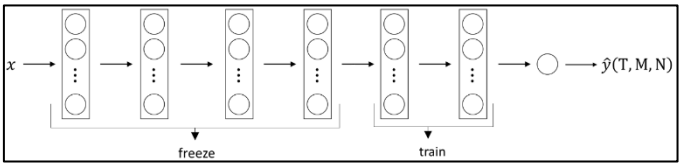
  - We will have our own SoftMax layer that outputs one of the three classes. as we are using downloaded weights, we will just train the parameters associated with our SoftMax layer



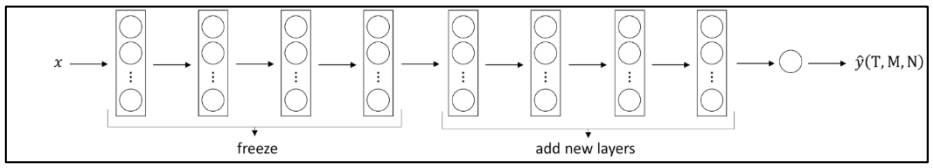
  - As early layers are frozen, there is some fixed function that doesn't change. We can pre compute & save feature activations from that layer. Then, we will train a shallow SoftMax model from this feature vector to make a prediction.



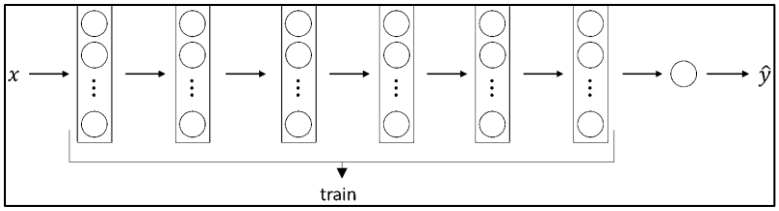
  - If we have a larger labelled dataset, we could freeze fewer layers then train these later layers.



- But if final layer outputs different number of classes then we need to have our own output unit. We can take the last few layer's weights and use that as an initialization so they would replace random initialization and do the gradient descent.
- We can also remove last few layer's weights, add new hidden units and final SoftMax output.



- If we have more data, smaller number of layers could be frozen and the number of layers to train on top could be greater. if we have a lot of data, we can take this open source network and weights, and use the whole architecture just as initialization and train the whole network.



- Transfer learning is something that we should almost always do, unless we have an exceptionally large dataset to train everything else from scratch by ourself.

Cases	Description	Application cases
Case1	Use VGG16& ImageNet as feature engineering (get CNN codes) from last layers & then train a logistic regression model on top of it	Image data is similar to ImageNet & dataset size is small
	Use VGG16& ImageNet as feature engineering (get CNN codes) after first 2 layers & then train a logistic regression model on top of it	Image data is dissimilar to ImageNet & dataset size is small
Case2	Fine tune the last few layers of vgg16	Image data is similar to ImageNet & dataset size is medium
	Retune the complete of vgg16	Image data is similar to ImageNet & dataset size is large
Case3	Use VGG16 + ImageNet as initial model & then tune the complete model keeping the learning rate low	Image data is dissimilar to ImageNet & dataset size is large
Case4	Learn everything from scratch	Image data is similar to ImageNet & dataset size is small